Table of Contents

Preparing Oracle Database for Profiling/Masking

Preparing SQL Server Database for Profiling and Masking

Preparing Sybase Database for Profiling and Masking

- Connecting Data

Managing Environments

Managing Remote Mounts

Managing Connectors

Managing Extended Connectors

Managing Rule Sets

Managing File Formats

Managing Inventories

Introduction

Introduction

- Identifying Sensitive Data

Discovering Your Sensitive Data

Out of the Box Profiling Settings

Managing Domains

Configuring Profiling Settings

Creating A Profiling Job

Running A Profiling Job

Reporting Profiling Results

- Securing Sensitive Data

- Introduction to Masking Algorithms
- dlpx-core:CM Alpha-Numeric
- dlpx-core:CM Digits
- dlpx-core:CM Numeric
- Credit Card
- Date Shift Discrete
- Date Shift Fixed
- Date Shift Variable
- dlpx-core:Email SL
- dlpx-core:Email Unique
- dlpx-core:FirstName

- runMaskingJob
- Authoring Extensible Plugins

Introduction

Introduction

Dependency Management

Plugin Metadata

Versioning

- Setting Up Your Development Environment
- Introduction
- The MaskingAlgorithm Java Interface
- Introduction
- Building the Sample Plugin
- Creating a New Project
- Service Discovery
- Running an Algorithm Using the SDK Tools
- Installing Multiple Plugins onto the Delphix Masking Engine
- Load Multiple Algorithm Plugins
- Retrieving Information about Installed Plugins
- Introduction
- Making an Algorithm Configurable
- Using an Algorithm Framework
- Using Multi-Column Algorithms
- Introduction
- Accessing Files
- Accessing Database Servers (JDBC)
- Algorithm Chaining
- Using Cryptographic Keys
- Logging
- Introduction
- Algorithm Implementation
- Introduction
- The DriverSupport Java Interface
- Introduction
- Building the Sample Plugin
- Creating a New Project
- Service Discovery
- Executing a Driver Support Task Using the SDK
- Retrieving Information about Installed Plugins
- Introduction
- Accessing Masking Engine Rulesets
- Accessing Database Servers (JDBC)
- Logging
- Managing Plugins Using the API Client

- [Installing a Plugin onto the Delphix Masking Engine](#)
- [Secure Plugin Deployment](#)
- [Terminology](#)

Introduction to Delphix Masking

Challenge

With data breach incidents regularly making the news and increasing pressure from regulatory bodies and consumers alike, organizations must protect sensitive data across the enterprise. Contending with insider and outsider threats while staying compliant with mandates such as HIPAA, PCI, and GDPR is no easy task—especially as teams simultaneously try to make their organizations more agile.

To tackle the problem of protecting sensitive information, companies are increasingly scrutinizing the tools they've deployed. Instead of reactive perimeter defenses, security-minded organizations must focus on proactively protecting the interior of their systems: their data. Moreover, while mainstay approaches such as encryption may be effective for securing data-in-motion or data resident in hard drives, they are ill-suited for protecting non-production environments for development, testing, and reporting.

Solution

The masking capability of the Delphix Dynamic Data Platform represents an automated approach to protecting non-production environments, replacing confidential information such as social security numbers, patient records, and credit card information with fictitious, yet realistic data.

Unlike encryption measures that can be bypassed through schemes to obtain user credentials, masking irreversibly protects data in downstream environments. Consistent masking of data while maintaining referential integrity across heterogeneous data sources enables Delphix masking to provide superior coverage compared to other solutions—all without the need for programming expertise. Moreover, the Delphix Dynamic Data Platform seamlessly integrates masking with data delivery capabilities, ensuring the security of sensitive data before it is made available for development and testing, or sent to an offsite data center or the public cloud.

Delphix Masking is a multi-user, browser-based web application that provides complete, secure, and scalable software for your sensitive data discovery, masking, and tokenization needs while meeting enterprise-class infrastructure requirements. The Delphix Dynamic Data Platform has several key characteristics to enable your organization to successfully protect sensitive data across the enterprise:

- **End-to-End Masking** — The Delphix platform automatically detects confidential information irreversibly masks data values, then generates reports and email notifications to confirm that all sensitive data has been masked.

- **Realistic Data** — Data masked with the Delphix platform is production-like in quality. Masked application data in non-production environments remain fully functional and realistic, enabling the development of higher-quality code.

- **Masking Integrated with Virtualization** — Most masking solutions fail due to the need for repeated, lengthy batch jobs for extracting and masking data and lack delivery capabilities for downstream environments. The Delphix Dynamic Data Platform seamlessly integrates data masking with data virtualization, allowing teams to quickly deliver masked, virtual data copies on premises or into private, public, and hybrid cloud environments.

- **Referential Integrity** — Delphix masks consistently across heterogeneous data sources. To do so, metadata and data is scanned to identify and preserve the primary/foreign key relationships between elements, so that data is masked the same way across different tables and databases.

- **Algorithms/Frameworks** — Seven algorithm frameworks allow users to create and configure algorithms to match specific security policies. Over twenty-five out-of-the-box, preconfigured algorithms help businesses mask everything from names and addresses to credit card numbers and text fields. Moreover, the Delphix platform includes prepackaged profiling sets for healthcare and financial information, as well as the ability to perform tokenization: a process that can be used to obfuscate data sent for processing, then reversed when the processed data set is returned.

- **Ease of Use** — With a single solution, Delphix customers can mask data across a variety of platforms. Moreover, businesses are not required to program their own masking algorithms or rely on extensive administrator involvement. Our web-based UI enables masking with a few mouse clicks and little training.

- **Automated discovery of sensitive data** — The Delphix Profiler automatically identifies sensitive data across databases and files, the time-consuming work associated with a data masking project is reduced significantly.

High-Level Platform Architecture

The Delphix Dynamic Data Platform is made up of 4 main services each of which play a very important part in delivering fresh secure data to anybody that needs it. These include:

- **Virtualize —** Delphix compresses the data that it gathers, often to one-third or more of the original size. From that compressed data footprint, Delphix virtualizes the data and allows operators to create lightweight, virtual data copies. Virtual copies are fully readable/writable and independent. They can be spun up or torn down in just minutes. And they take up a fraction of the storage space of physical copies -- 10 virtual copies can fit into the space of one physical copy.

- **Identify and Secure —** The Delphix platform continuously protects sensitive information with integrated data masking. Masking secures confidential data -- names, email addresses, patient records, SSNs -- by replacing sensitive values with fictitious, yet realistic equivalents. Delphix automatically identifies sensitive values then applies custom or predefined masking algorithms. By seamlessly integrating data masking and provisioning into a single platform, Delphix ensures that secure data delivery is effortless and repeatable.

- **Manage —** Data operators can now quickly provision secure data copies -- in minutes -- to users in their target environments. The Delphix platform serves as a single point of control to manage those copies. Data operators maintain full control and visibility into downstream environments. They can easily audit, monitor, and report against access and usage.

- **Self Service —** Provides developers, testers, analysts, data scientists, or other users with controls to manipulate data at-will. Users can refresh data to reflect the latest state of production, rewind environments to a prior point in time, bookmark data copies for later use, branch data copies to work across multiple releases, or easily share data with other users.
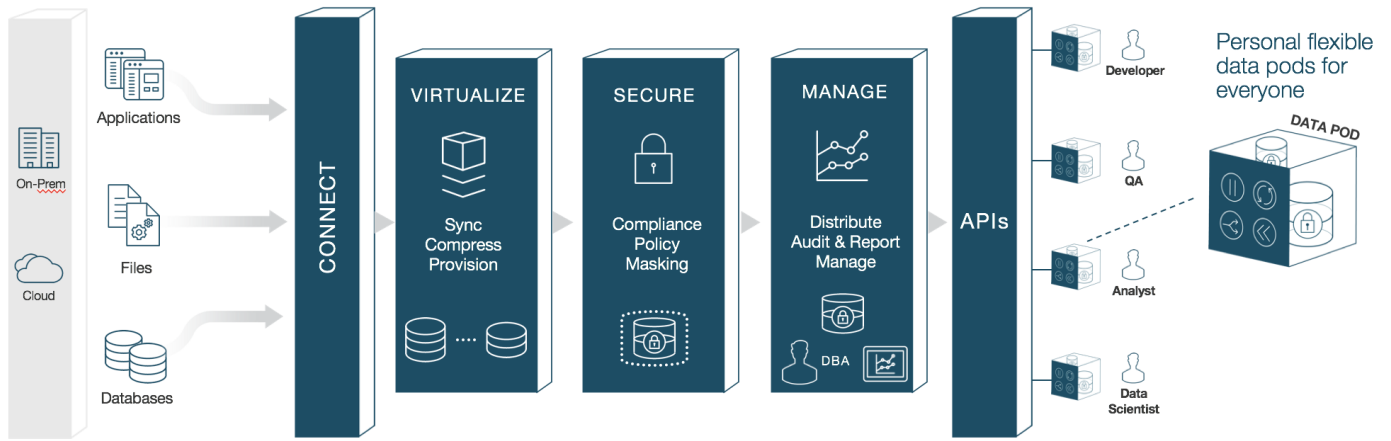
How Delphix Identifies Sensitive Data

Our platform helps you quickly identify your organization's sensitive data. This sensitive data identification is done using two different methods, column-level profiling, and data level profiling.

Column Level Profiling

Column level profiling uses REGEX expressions to scan the column names (metadata) of the selected data sources. There are several dozen pre-configured profile expressions (like the one below) designed to identify common sensitive data types (SSN, Name, Addresses, etc). You also have the ability to write/import your own profile expressions.

First Name Expression	<([A-Z][A-Z0-9]*)\b[^>]*>(.*?)</\1>

Data Level Profiling

Data level profiling also uses REGEX expressions, but to scan the actual data instead of the metadata. Similar to column level profiling, there are several dozen pre-configured expressions (like the one below) and you can write/import your own.

Social Security Number Expression	<([A-Z][A-Z0-9]*)\b[^>]*>(.*?)</\1>

For both column and data level profiling, when data is identified as sensitive, Delphix recommends/assigns particular algorithms to be used when securing the data. The platform comes with several dozen pre-configured algorithms which are recommended when the profiler finds certain sensitive data.

How Delphix Secures Your Sensitive Data

Delphix strives to make available multiple methods for securing your data, depending on your needs. The two secure methods Delphix currently supports are masking (anonymization) and tokenization (pseudonymization).

Masking

Data masking secures your data by replacing values with realistic yet fictitious data. Seven out-of-the-box algorithm frameworks help businesses mask everything from names and social security numbers to images and text fields. Algorithms can also be configured or customized to match specific security policies.

Before Masking	After Masking
Elon Musk	Jeff Bezos

Tokenization

Tokenization uses reversible algorithms so that the data can be returned to its original state. Tokenization is a form of encryption where the actual data – such as names and addresses – are converted into tokens that do not convey any meaning (with regards to appearance and formatting).

Before Tokenizing	After Tokenizing
226-74-3756	asdflkajsfdaja

What's New for Masking

6.0.9.0 Release

Masking SDK Driver Support Plugins

The Masking SDK functionality is extended with the ability to develop a new kind of plugin called driver support plugins that allow the execution of developer-defined tasks as part of a masking job.

Masking SFTP Connector is extended with a new flag UserDirIsRoot

Delphix introduces a new flag, setting whether the SFTP Connector configured Path is relative or absolute.

New Email Framework

Delphix introduces a new Email Framework along with two default algorithm instances. This functionality allows for more customization in masking email addresses.

New Copy Environment API

In this release, Delphix has extended the list of API-endpoints by adding a new API for copying environments.

The API endpoint is :

Group	Endpoints	Description
environment	POST /environments/{environmentId}/copy	Copy environment objects in the same or a different application

6.0.8.0 Release

New Name and Full Name Frameworks

Delphix introduces new Name and Full Name Frameworks, as well as their default algorithms instances. That functionality adds flexibility and more sophisticated way for names masking.

Masking SDK multiple plugins capacity

Masking SDK functionality is extended with an option of loading multiple plugins and chaining extensible algorithms based on different plugins. The default dlpx-core plugin is uploaded by default.

New Regex Decompose Algorithm Chaining Framework

Delphix introduces the Regex Decompose extensible algorithm framework which allows building new algorithms from a combination of predefined actions and existing algorithms.

Enclosure escape character support for delimited file masking

In this release, Delphix has added the escape character support for delimited file masking. Specifically the following were added:

Enclosure Escaping Strategy

The user can configure the enclosure escape character from the UI/API to escape the enclosure. To configure the enclosure escape character from the UI, user needs to select the "Enclosure Escaping Strategy" dropdown value as per the below options on the edit Rule Set popup window,

1. **Double Enclosure:** Double enclosure option will set the escape character value same as enclosure value.
2. **Custom:** By selecting custom option user can specify any single character as an enclosure escape character except the "escape sequences" and "control characters".

Escape "Enclosure Escape Character"

The user can escape the "enclosure escape character" itself by clicking on the Escape "Enclosure Escape Character" checkbox on the edit RuleSet popup window.

For more detailed information click here.

6.0.7.0 Release

New Date Masking Frameworks

Delphix introduces new date masking frameworks including date replacement, date shift, and multi-column dates. These new frameworks obviate the need for many of the custom date algorithms that were required in the past. Delphix also introduces new default implementations of common date-masking functionality. The new date masking frameworks are briefly described below.

- Date Replacement: Selects a replacement value from a customer configurable date range.
- Date Shift: Produces a replacement value by randomly shifting the input date by a customer configurable increment range.
- Multi-column Date: Masks date values that have a dependency, such as admission and discharge date using the same algorithm as Date Shift. This allows masking of both the initial date and the difference between the dates.

New Credit Card Masking Algorithms

Delphix introduces a robust payment-card masking framework as well as a default algorithm implementation for credit card data. The legacy credit card algorithm, which produced random values, is being replaced by the new default instance which provides consistent masking results, a unique output for every valid input, always changes a valid input value, and preserves all non-digit portions of the input value.

Masking Engine changes for Users and Groups

This enhancement adds stronger on-Masking Engine safeguards to the users and group's experience delivered in Central Management in which the access to a Masking Engine's objects is determined by assigning authorization via global access groups. Specifically, when an engine opts into the global model, it relinquishes local control of object access. With this enhancement, the local enforcement of global (Central Management) settings is strengthened by deactivating local object access in the UI, thus ensuring the local values will not be overridden via frequent, periodic scans from Central Management.

New Forgot and Reset password APIs

In this release, Delphix has extended the list of API-endpoints by adding two new API's related to the existing Forgot and Reset password feature for a user, which was available via GUI only till now.

The two new sets of API endpoints are :

Group	Endpoints	Description
user	POST /users/forgot-password	Send reset password mail to the user
	POST /users/reset-password	Reset new password for the user

The forgot-password API will generate and send a password reset link to the registered email id of the user, for which the password has to be reset.

The reset-password API will use the token sent via the password reset link, to set the new password.

Control character support for delimited file masking

In this release, Delphix has added the control character support for delimited file masking. Specifically the following were added:

1. **Control character as a delimiter:** The user can specify a control character as the delimiter from UI/API.
2. **Control character as an end of record:** The user can specify a control character as the end of record from UI/API.
3. **Control character as a value:** Delimited files containing values with control characters are now supported.

Date-Time format change for the API response

In this release, the date-time format for API responses is changed

From: yyyy-MM-dd'T'HH:mm:ss.SSSZ *e.g. 2021-03-17T17:35:39.352+0000*

To: yyyy-MM-dd'T'HH:mm:ss.SSSXXX *e.g. 2021-03-17T17: 35:39.352+00:00*.

The API endpoints below will be affected by this change:

- GET /system-information
- GET /plugin
- GET /profile-jobs
- GET /profile-sets
- GET /execution-events
- GET /async-tasks
- GET /audit-logs
- GET /algorithms in algorithm extension object
- GET /execution-components
- GET /jdbc-drivers
- GET /masking-jobs
- GET /reidentification-jobs
- GET /tokenization-jobs

6.0.6.0 Release

Multi-Column Algorithm

In this release, Delphix has introduced a Multi-Column Extensible Algorithm mechanism, which allows masking multiple columns of the same table conditional to their values (or using any other logic needed by the customer). To use the Multi-Column Algorithm Framework, users first create an algorithm via the Masking SDK and then install their algorithm on a Masking Engine via the Extensible Algorithm Plugin interface.

Latest Api Version

The latest masking API version supported on the engine will be included in the `GET /system-information` API response.

Custom Database Connection Properties

There is now a way to specify custom connection properties for all of our database connector types by uploading a properties file. See Database Connection Properties for more info.

Certifications

- DB2 iSeries v7.4

6.0.5.0 Release

Character Mapping Algorithm

Delphix is introducing a replacement for the Segment Mapping Algorithm, the Character Mapping Algorithm. The new Character Mapping Algorithm is built using the recently released algorithm SDK, and in most common configurations this new algorithm will be faster and require less memory than the existing segment mapping algorithm. In addition, this new version does not have a length limitation for the input string and can handle non-ASCII characters.

Certifications

- MySQL 8

- Postgres SQL 12

- DB2 LUW 11.5

- Oracle Database Cloud Services on Virtual Machines

- Oracle Database Cloud Services on Bare Metal

- Google Cloud SQL for PostgreSQL

- Google Cloud SQL for MySQL

- Google Cloud SQL for SQL Server

Default Api Version

Ability to specify the Masking API version to be used when the version is omitted from the base path of the Masking API request's URL.

New API Version

To reflect the API improvements mentioned above, the API version increased to 5.1.5 in this release. For a complete listing of version 5.1.5, see the Masking API Client page.

6.0.4.0 Release

Masking Job Memory Improvements

Memory management has been dramatically improved. Not only can jobs run with less memory, but the Masking Engine will also now ensure that jobs can only run if enough memory is available and that the engine cannot run out of memory.

Along with these changes, there are two new execution statuses: `CANCELLED` and `QUEUED`.

Extensible Connector Permissions Change

The first iteration of the Masking Extensible Connectors, supporting the ability to upload and use JDBC drivers, required that the permissions for each driver be enumerated at install time. Delphix has now replaced this mechanism with a fixed security policy blocking only the most dangerous permissions (specifically those that could inflict harm to the Masking Engine), removing the need for user management of permissions. It remains the case that the engine administrator must ensure that only trusted JDBC driver software is installed.

File Masking Performance

The performance of file masking has been significantly improved.

Builtin Extensible Secure Lookup Framework

Delphix has added a builtin, configurable Secure Lookup Algorithm Framework, based on the Extensible Algorithms feature (introduced in 6.0.3.0 release).

This framework provides better performance and new features when compared with the Legacy Secure Lookup Algorithms.

It allows configuring the case sensitivity of input values (true/false), and the case configuration of the output values:

```
Preserve Lookup File Case    // i.e. as found in Lookup File
Preserve Input Case          // i.e. preserve case of input value - UpperCase / LowerCase / Mixed
Force all Lowercase          // forces output to LowerCase
Force all Uppercase          // forces output to UpperCase
```

The algorithm instance (based on the new Secure Lookup Algorithm Framework) might be managed via the existing Algorithm API, similar to any other plugin algorithm. The GUI has been changed for configuring/editing Secure Lookup Algorithm. For details please see the Secure Lookup Algorithm Framework

Job Scheduler Removed

As of this release, we have removed the Job Scheduler feature. The introduction of Masking's REST API several releases ago allowed customers to schedule job executions using their preferred job scheduler. As a result, the integrated scheduler is seldom used.

Certifications

This release adds support for SQL Server 2017 and 2019.

Free Text Redaction Algorithm

The redaction strategies used in a free text redaction algorithm have been renamed to "Allowlist" and "Denylist".

New API Version

To reflect the API improvements mentioned above, the API version increased to 5.1.4 in this release. For a complete listing of version 5.1.4, see the Masking API Client page.

6.0.3.0 Release

Extensible Algorithms

We introduced a new, radically simpler, method to create new masking algorithms. With the new framework, Delphix partners and customers can create and share new algorithms.

Extensible algorithms and their related algorithm plugins can be managed through the following APIs:

Group	Endpoints	Description
plugin	GET /plugin	Get all plugins
	POST /plugin	Install plugin
	DELETE /plugin/{pluginId}	Delete plugin
	GET /plugin/{pluginId}	Get plugin detail by pluginId
	PUT /plugin/{pluginId}	Update plugin

Existing *algorithm* API is extended with the following endpoints:

Group	Endpoints	Description
algorithm	GET /algorithm/frameworks	Get all algorithm frameworks
	GET /algorithm/frameworks/id/{frameworkId}	Get algorithm framework by frameworkId

UI-based Environment Sync

Over the past several releases Delphix has introduced and refined the ability to synchronize objects between Masking Engines via the API. In 6.0.3, Delphix now supports importing and exporting environments via the UI.

> ✏️ **Note**
>
> In this release, the deprecated XML import/export functionality has been removed. If you used the XML import/export feature in previous releases, you'll find the new Sync Environment feature to be a more robust and complete solution with complete API support in addition to being available in the UI.

New SQL Server JDBC Driver

The product switched from the jTDS JDBC driver to Microsoft's official open-source JDBC driver. This was done to obtain improved support for recent versions of SQL Server.

All SQL Server basic connectors will be converted transparently. If you used a SQL Server Advanced connector or a Generic connector using the jTDS driver, you will need to manually convert your JDBC URL to the Microsoft JDBC driver's format. To perform this conversion, please see the references for the jTDS parameters and the Microsoft JDBC parameters. Delphix Customer Support's upgrade validation checks will detect any SQL Server Advanced connectors and Generic connectors using the jTDS driver in your installation and they will notify you of the need to manually convert those connectors.

AzureSQL Managed Databases

This release is certified to be compatible with the following Azure SQL Managed Databases:

- Azure Database for PostgreSQL service
- Azure Database for MySQL service
- Azure Database for MariaDB service
- Azure Database for SQL

> ✏ **Note**
>
> You must enable support for non-TLS connections.

File Masking Performance

This release contains significant performance improvements for delimited and XML file masking.

New API Version

To reflect the API improvements mentioned above, the API version increased to 5.1.3 in this release. For a complete listing of version 5.1.3, see the Masking API Client page.

6.0.2.0 Release

Mainframe Data Set Improvements for Masking

This release delivers multiple quality-of-experience enhancements around mainframe masking workflows:

- **Mainframe Masking Performance:** Anyone masking mainframe data sets may see a large improvement in performance.
- **Engine Sync Support for Mainframe:** The Sync APIs and workflows now support mainframe objects: connectors, rule sets, jobs, and formats.

- **Mainframe Data Set Record Type APIs:** This enhancement builds upon the recent release of Record Type APIs to include mainframe support. You will now be able to manage Mainframe data set record types via REST API, including redefine conditions. When masking a mainframe data set, the Masking Engine uses a mainframe data set format to interpret the data set's contents. A mainframe data set format has one default record type "All Record". If a mainframe data set format contains redefined fields, each redefined and redefines field will have a corresponding record type that holds the redefined condition for the redefined and redefines fields. Specifically, the following APIs were added:

Group	Endpoints	Description
mainframeDatasetRecordType	GET /mainframe-dataset-record-types	Get all Mainframe Dataset record type
	GET /mainframe-dataset-record-types/{mainframeDatasetRecordTypeId}	Get Mainframe Dataset record type by ID
	PUT /mainframe-dataset-record-types/{mainframeDatasetRecordTypeId}	Update Mainframe Dataset record type by ID

For more information on redefine conditions, see the Managing a Mainframe Inventory section.

JDBC to Delimited Files Support

On-the-fly masking jobs with a JDBC source and delimited file target are now supported. This is targeted at users with data lake applications. This is targeted at users with data lake applications who wish to extract unmasked data using a JDBC connection and insert masked data back using a bulk file load mechanism.

Environment Sync Support for Masking

With this release, an entire environment is now syncable with a single operation via the Sync REST APIs. Previously, Sync users would have to export/import objects on an individual basis, the process now is far more streamlined. Note: Environment Sync APIs are the preferred way of handling environment export/import versus XML-based transfer.

New API Version

To reflect the API improvements mentioned above, the API version increased to 5.1.2 in this release. For a complete listing of version 5.1.2, see the Masking API Client page.

Certifications

This release adds support for Oracle 19c.

6.0.1.0 Release

Extended Connectors

Extended Connectors is a new feature that allows you to upload additional JDBC Drivers to the Delphix Masking engine. This enables masking data sources that are not natively supported by Delphix Masking. For more information, please refer to the Managing Extended Connectors section.

Sync for Tokenization and Reidentification Jobs

The Sync feature allows you to coordinate the operation of multiple engines. This release adds Sync support for Tokenization and Reidentification Jobs. For more information on the Sync feature, please refer to the Managing Multiple Engines for Masking section.

File Record Type APIs

When masking a delimited or fixed length file, the Masking Engine uses a file format to interpret the file's contents. Each format has one or more record types. In previous releases, these record types could only be created and managed through the graphical user interface. This release adds the ability to also create and manage file record types through the APIs. Specifically, the following APIs were added:

Group	Endpoints	Description
recordType	GET /record-types	Get all record type
	POST /record-types	Create record type
	DELETE /record-types/{recordTypeId}	Delete record type by ID
	GET /record-types/{recordTypeId}	Get record type by ID
	PUT /record-types/{recordTypeId}	Update record type
recordTypeQualifier	GET /record-type-qualifiers	Get all record type qualifiers
	POST /record-type-qualifiers	Create record type qualifier
	DELETE /record-type-qualifiers/{recordTypeQualifierId}	Delete record type qualifier by ID
	GET /record-type-qualifiers/{recordTypeQualifierId}	Get record type qualifier by ID
	PUT /record-type-qualifiers/{recordTypeQualifierId}	Update record type qualifier by ID

Note that record types are only used for delimited and fixed length file formats. For more information on record types, see the Adding Record Types for Files section.

6.0.0.0 Release

Objects Names Requirements

In 6.0 we have added validations for objects names that can be created/renamed manually. For more information please refer to Naming Requirements. Please pay attention to the fact that enforcing these requirements might fail the import, sync, or upgrade from pre-6.0 release. Please refer to the following Knowledge Base Article KBA5096 on how to solve those failures.

Versioning Framework

6.0 marks the release of version 5.1 of the Masking API. For information on how the Masking API is versioned, please refer to the documentation here: Masking API Versioning Documentation

New API Endpoints

In 6.0 we have expanded the list of API endpoints to include:

Group	Endpoints	Description
Application	DELETE /applications/{applicationId}	Delete application by ID
Mount Filesystem	GET /mount-filesystem	Get all mounts
	POST /mount-filesystem	Create a mount
	GET /mount-filesystem/{mountId}	Get a mount by ID
	DELETE /mount-filesystem/{mountId}	Delete a mount by ID
	PUT /mount-filesystem/{mountId}	Update a mount by ID
	PUT /mount-filesystem/{mountId}/connect	Connect a mount by ID
	PUT /mount-filesystem/{mountId}/disconnect	Disconnect a mount by ID
	PUT /mount-filesystem/{mountId}/remount	Remount a mount by ID

In addition to the new API endpoints, we have improved existing API endpoints. These improvements include:

- Addition of the applicationId field to the application model

- Replacement of the application field with an applicationId field in the Environment model

- Removal of the classification field from the domain model

- Addition of the rulesetType field to the Masking, Profiling, Reidentification, and Tokenization job models.

- Addition of mountName in the ConnectionInfo of a file connector and a mainframe dataset connector to use a filesystem mount point.

For more information on Delphix Masking APIs please see the [API documentation](#).

NFS and CIFS Mounts

In previous releases, the Masking Engine has supported masking files via FTP or SFTP. In this release, we have added the ability for users to directly mount and mask a file system over NFS and CIFS. This should dramatically simplify the process of file masking. As with other Masking Engine objects, the Sync feature can be used to coordinate mount objects across multiple engines. For more information on the mount feature, please refer to the [Managing Remote Mounts](#) section.

5.3 Release

Synchronizing Masking Jobs and Universal Settings Across Engines

In 5.2 we introduced the ability to synchronize Masking Algorithms between engines to ensure consistent masking, regardless of the engine executing the masking. In 5.3 we are expanding the list of syncable objects to include:

- Masking Jobs

- Connectors

- Rulesets

- Domains

- File Formats

The sync of objects is possible through improvements to several sync API endpoints, including:

- GET /syncable-objects[?object_type=]

- POST /export

- POST /export-async

- POST /import

- POST/import-async

This expansion of syncable objects ensures that users can sync their Masking Jobs and all the objects necessary for that masking job to execute successfully - regardless of the masking engine it lives on, allowing for easier scaling of Delphix Masking across the enterprise. Please see [Managing Multiple Masking Engines](#) for more details.

Support for Kerberized Connections

In 5.2.4 we added support for Kerberos for our Oracle Masking Connector. In 5.3 we have expanded the list of connectors that support Kerberos to:

- SQL Server

- Sybase

To enable Kerberized connectors your engine must be configured properly and you must configure your masking Connectors for Kerberos. Kerberos can be enabled by going to the Advanced mode on Oracle, SQL Server and Sybase. Please see Managing Connectors for more details.

New API Endpoints

In 5.2 we released an all-new set of API endpoints allowing for the automation of many masking workflows. In 5.3 we have expanded this list of API endpoints around Algorithms, Users, Roles, File Upload, System Information, Login, Rulesets, and Connector. Below are the net new API endpoints:

Group	Endpoints	Description
Algorithms	POST /algorithms	Create algorithm
	DELETE /algorithms/{algorithmName}	Delete algorithm by name
	GET /algorithms/{algorithmName}	Get algorithm by name
	PUT /algorithms/{algorithmName}	Update algorithm by name
	PUT /algorithms/{algorithmName}/randomize-key	Randomize key by name

Group	Endpoints	Description
Users	GET /users	Get all users
	POST /users	Create user
	DELETE /users/{userId}	Delete user by ID
	GET /users/{userId}	Get user by ID
	PUT /users/{userId}	Update user by ID
Roles	GET /roles	Get all roles
	POST /roles	Create role
	DELETE /roles/{roleId}	Delete role by ID
	GET /roles/{roleId}	Get role by ID
	PUT /roles/{roleId}	Update role by ID
Rulesets	PUT /database-rulesets/{databaseRulesetId}/bulk-table-update	Update the rule set's tables
	PUT /database-rulesets/{databaseRulesetId}/refresh	Refresh the rule set
Connectors	POST /database-connectors/{databaseConnectorId}/test	Test a database connector
	POST /database-connectors/test	Test an unsaved database connector
	POST /file-connectors/{fileConnectorId}/test	Test a file connector
	POST /file-connectors/test	Test an unsaved file connector
Async Tasks	GET /async-tasks	Get all asyncTasks
	GET /async-tasks/{asyncTaskId}	Get asyncTask by ID
	PUT /async-tasks/{asyncTaskId}/cancel	Cancel asyncTask by ID

Group	Endpoints	Description
File Upload/Download	DELETE /file-uploads	Delete all file uploads
	POST /file-uploads	Upload file
	GET /file-downloads/{fileDownloadId}	Download file
System Information	GET /system-information	Get version, etc.
Login/Logout	PUT /logout	User logout
Executions	GET /execution-components	Status for a table, file, or Mainframe data set
Tokenization Job	GET /tokenization-jobs	Get all tokenization jobs
	POST /tokenization-jobs	Create tokenization job
	DELETE /tokenization-jobs/{tokenizationJobid}	Delete tokenization job by ID
	GET /tokenization-jobs/{tokenizationJobid}	Get tokenization job by ID
	PUT /tokenization-jobs/{tokenizationJobid}	Update tokenization job by ID
Re-identification Job	GET /reidentification-jobs	Get all re-identification jobs
	POST /reidentification-jobs	Create re-identification job
	DELETE /reidentification-jobs/{reidentificationJobid}	Delete re-identification job by ID
	GET /reidentification-jobs/{reidentificationJobid}	Get re-identification job by ID
	PUT /reidentification-jobs/{reidentificationJobid}	Update re-identification job by ID
Database Rulesets	PUT	Update Database Ruleset by ID

In addition to the net new API endpoints, we have improved pre-existing API endpoints. Some of the improvements include:

- Addition of DB2 iSeries and Mainframe to connector endpoints.

- Addition of Kerberos configuration on Oracle, SQL Server, and Sybase connectors

- Ability to have ruleset refresh drop tables

- Support for XML file types

- Addition of dataType to column metadata

- Addition of isProfilerWritable field to file-field-metadata endpoints. This is now represented in the API as a new *isProfilerWritable* boolean field in the body of a file-field-metadata. When the isProfilerWritable field is set to true, the algorithm/domain assignment on a column can be overwritten by the profiler. When the field is false, it may not be overwritten.

- Addition of multipleProfilerCheck field to Profile Job endpoints. This feature is turned on using the boolean field in the body of a profile job. The job profiler normally stops profiling a column as soon as it flags a field as sensitive. If *multipleProfilerCheck* is true, the profiler will continue to scan the column for additional sensitive patterns. In the event that it finds more than one pattern, it will tag all the data domains found and apply 'one' standard algorithm for all those domains. The standard algorithm is 'Null SL' as of 5.3.4.0. This feature was formerly called 'multi PHI'.

For more information on Delphix Masking APIs please see the API documentation. Please note that the previous generation of Masking APIs (commonly referred to as V4) is EOL and no longer supported in this release. All users are encouraged to migrate to the V5 APIs.

Deprecated and Removed Features

Release 6.0.9.0

End of Life Notifications

Delphix has been creating new and improved versions of our existing algorithms, thus, Delphix would like to provide formal notice of deprecation and planned End-of-Life (EoL) for the older algorithm versions. This is to inform our customers that planning should start for their transition to these updated algorithms.

Details of the transition can be found in the following Delphix Community Post.

Release 6.0.8.0

End of Life Notifications

- Legacy Custom Algorithm (Mapplet) end-of-life. More information available in this Delphix Community Post.
- SAP ASE (Sybase) 15.0.3 support end-of-life in 6.0.8

Release 6.0.7.0

End of Life Notifications

- ESX 5.5 support will be end-of-life in 6.0.7
- Masking Connectors: Db2 LUW and zOS v9, Db2 LUW and zOS v10, SQL Server 2005, 2008, 2008 R2

Release 6.0.4.0

The following features were removed in the 6.0.4 release:

- Job Scheduler - As of this release, Delphix has removed the Job Scheduler feature. The introduction of Masking's REST API several releases ago allowed customers to schedule job executions using their preferred job scheduler. As a result, the integrated scheduler is seldom used.

The following features are deprecated as of 6.0.4 and will be removed in future releases:

- FTP, SFTP, and mount upload for XML and Cobol formats - FTP/SFTP/Mount-based format import were the original modes for XML and Cobol files, since then, Delphix has added the ability to upload a format file, which is far simpler to set up. After the introduction of "upload", we've seen a dramatic shift away from the legacy import modes in favor of the simplicity of "upload".
- Row Type Feature - Originally geared for limiting masking to subsets of rows within a column, this feature was seldomly used. Its functionality, if desired, can still be replicated via the Custom SQL feature.

- Redundant Settings for 'Edit Table' under Rule Sets - Table Suffix, Add Column, Join Table, and List - These settings are redundant and can be replicated with the Custom SQL setting.
- 'HAVING' clause from Masking API - Deprecating due to low use. This feature, if desired, can be replicated with Custom SQL.

Release 6.0.3.0

Removed Features

In this release, the deprecated XML import/export functionality has been removed. If you used the XML import/export feature in previous releases, you'll find the new Sync Environment feature to be a more robust and complete solution with complete API support in addition to being available in the UI.

Release 6.0.0.0

Deprecated Features

In this release the following features were deprecated:

Removed Masking Features: Please note that Excel files can still be masked by first converting them to one of Delphix's supported file types (CSV, etc). Also, XML CLOBs can be masked by extracting their values into a table (example - using extractValue in Oracle).

- Native XML CLOB masking: After upgrade, columns masked as XML CLOBs will have the NULL SL algorithm assigned.
- DB2 9.1, 9.5, and other 9.x versions of LUW & Z/OS
- "Create target" job option: After upgrade jobs using "create target" will be removed.
- "Bulk data" job option: After upgrade, jobs using "bulk data" will be turned into non-bulk data jobs.
- Native Microsoft Excel Masking: After upgrade, MS Excel connectors, rulesets and jobs will be removed.

Getting Started

Data Source Support

The Delphix Masking service supports profiling, masking, and tokenizing a variety of different data sources including distributed databases, mainframe, PaaS databases, and files. At a high level, Delphix Masking breaks up support for data sources into two categories:

- **Dedicated Delphix Connectors:** These are data sources that the Delphix Engine can connect to directly using built-in connectors that have been optimized to perform masking, profiling and tokenization.
- **FEML Sources:** FEML (File Extract Mask and Load) is a method used to mask and tokenize data sources that do not have dedicated Delphix Connectors. FEML uses existing APIs from data sources to extract the data to a file, masks the file, and then uses APIs to load the masked file back into the database.

Dedicated Delphix Connectors

The Delphix Engine has dedicated masking connectors for the following data sources:

- **Distributed Database:** DB2 LUW, Oracle, MS SQL, MySQL, SAP ASE (Sybase), PostgreSQL, MariaDB
- **Mainframe/Midrange:** DB2 Z/OS, DB2 iSeries, Mainframe data sets
- **Files:** Fixed Width, Delimited, XML

For a detailed view of all the versions, features, etc Delphix supports on each data source - see the sections below.

DB2 LUW Connector

Introduction

DB2 for Linux, UNIX, and Windows is a database server product developed by IBM. Sometimes called DB2 LUW for brevity, it is part of the DB2 family of database products. DB2 LUW is the "Common Server" product member of the DB2 family, designed to run on the most popular operating systems. By contrast, all other DB2 products are specific to a single platform.

Support Matrix

| **Platforms** | | ----------- | | **Unix** | | **Linux** | | **Windows** |

| **Versions** | | ----------- | | **11.1** | | **11.5** |

| | **Feature** | **Availability** | | ------------------------ | -------------------- | ---------------- | | **In-Place Masking Mode** | Multi-Tenant | Available | | | Streams / Threads | Available | | | Batch Update | Available | | | Drop Indexes | Available | | | Disable Trigger | Unavailable | | | Disable Constraint | Unavailable | | | Identity Column Support | Unavailable | | **On-The-Fly Masking Mode** | Restart Ability | Available | | | Truncate | Available | | | Disable Trigger | Unavailable | | | Disable Constraint | Unavailable | | **Profiling** | Multi-Tenant | Available | | | Streams | Available |

Oracle Connector

Introduction

Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is a multi-model database management system produced and marketed by Oracle Corporation.

Support Matrix

| **Platforms** | | | -------- --- | | **Unix** | | **Linux** | | **Windows** | | **AWS RDS** | |**OCI DBaaS on Bare Metal**| |**OCI DBaaS on VM**|

| **Versions** | | | ----- ------ | | **10g** | | **11gR1** | | **11gR2** | | **12c** | | **12cR2** | | **18c** | | **19c** |

| | **Feature** | **Availability** | | ------------------------ | --------- ------------ | ---------------- | | **In-Place Masking Mode** | Multi-Tenant | Available | | | Streams / Threads | Available | | | Batch Update | Available | | | Drop Indexes | Available | | | Disable Trigger | Available | | | Disable Constraint | Available | | | Identity Column Support | Available | | **On-The-Fly Masking Mode** | Restart Ability | Available | | | Truncate | Available | | | Disable Trigger | Available | | | Disable Constraint | Available | | **Profiling** | Multi-Tenant | Available | | | Streams | Available |

MS SQL Connector

Introduction

Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet).

Support Matrix

| **Platforms** | | | -------- --- | | **Unix** | | **Linux** | | **Windows** | | **AWS RDS** | | **Azure SQL** | |**Google Cloud SQL Server**|

| **Versions** | | | --- -------- | | **2012** | | **2014** | | **2016** | | **2017** | | **2019** |

| | **Feature** | **Availability** | | ------------------------ | ----------- ---------- | ------------- | | **In-Place Masking Mode** | Multi-Tenant | Available | | | Streams / Threads | Available | | | Batch Update | Available | | | Drop Indexes | Available | | | Disable Trigger | Available | | | Disable Constraint | Available | | | Identity Column Support | Available | | **On-The-Fly Masking Mode** | Restart Ability | Available | | | Truncate | Available | | | Disable Trigger | Available | | | Disable Constraint | Available | | **Profiling** | Multi-Tenant | Available | | | Streams | Available |

PostgreSQL Connector

Introduction

PostgreSQL, often simply Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance. PostgreSQL is developed by the PostgreSQL Global Development Group, a diverse group of many companies and individual contributors. It is free and open-source, released under the terms of the PostgreSQL License, a permissive software license.

Support Matrix

| **Platforms** | | ---------- | | **Unix** | | **Linux** | | **Windows** | | **AWS RDS** | | **AWS Aurora** | | **Azure Database for PostgresSQL** | | **Google Cloud SQL PostgresSQL**|

| **Versions** | | ------------ | | **9.2** | | **9.3** | | **9.4** | | **9.5** | | **9.6** | | **10** | | **11** | | **12** |

| | **Feature** | **Availability** | ------------------------ | ---------------------- | ---------------- | | **In-Place Masking Mode** | Multi-Tenant | Available | | | Streams / Threads | Available | | | Batch Update | Available | | | Drop Indexes | Unavailable | | | Disable Trigger | Unavailable | | | Disable Constraint | Unavailable | | | Identity Column Support | Available | | **On-The-Fly Masking Mode** | Restart Ability | Unavailable | | | Truncate | Available | | | Disable Trigger | Available | | | Disable Constraint | Available | | **Profiling** | Multi-Tenant | Available | | | Streams | Unavailable |

MySQL / MariaDB Connector

Introduction

MySQL is an open-source relational database management system (RDBMS). MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB. MySQL is now owned by Oracle Corporation.

MariaDB is a community-developed fork of the MySQL relational database management system intended to remain free under the GNU GPL. Development is led by some of the original developers of MySQL, who forked it due to concerns over its acquisition by Oracle Corporation.

A MySQL Connector may be used to connect to either a MySQL or MariaDB database instance.

MySQL Support Matrix

| **Platforms** | | ---------- | | **Unix** | | **Linux** | | **Windows** | | **AWS RDS** | | **AWS Aurora** | | **Azure Database for MySQL** | | **Google Cloud SQL MySQL**|

| **Versions** | | ----------- | | **5.5** | | **5.6** | | **5.7** | | **8** |

| | **Feature** | **Availability** | | ------------------------ | ---------------------- | ---------------- | | **In-Place Masking Mode** | Multi-Tenant | Available | | | Streams / Threads | Available | | | Batch Update | Available | | | Drop Indexes | Available | | | Disable Trigger | Unavailable | | | Disable Constraint | Unavailable | | | Identity Column Support | Available | | **On-The-Fly Masking Mode** | Restart Ability | Unavailable | | | Truncate | Available | | | Disable Trigger | Unavailable | | | Disable Constraint | Unavailable | | **Profiling** | Multi-Tenant | Available | | | Streams | Available |

MariaDB Support Matrix

| **Platforms** | | --------- -- | | **Unix** | | **Linux** | | **Windows** | | **AWS RDS** | | **AWS Aurora** | | **Azure Database for MariaDB** |

| **Versions** | | ---------- | | **10** |

| | **Feature** | **Availability** | | ----------------------- | ---------------- | ---------------- | | **In-Place Masking Mode** | Multi-Tenant | Available | | | Streams / Threads | Available | | | Batch Update | Available | | | Drop Indexes | Available | | | Disable Trigger | Unavailable | | | Disable Constraint | Unavailable | | | Identity Column Support | Available | | **On-The-Fly Masking Mode** | Restart Ability | Unavailable | | | Truncate | Available | | | Disable Trigger | Unavailable | | | Disable Constraint | Unavailable | | **Profiling** | Multi-Tenant | Available | | | Streams | Available |

SAP ASE (Sybase) Connector

Introduction

SAP ASE (Adaptive Server Enterprise), originally known as Sybase SQL Server, and also commonly known as Sybase DB or Sybase ASE, is a relational model database server product for businesses developed by Sybase Corporation which became part of SAP AG.

Support Matrix

| **Platforms** | | -- --------- | | **Unix** | | **Linux** | | **Windows** |

| **Versions** | | - ---------- | | **15.5** | | **15.7** | | **16** |

| | **Feature** | **Availability** | | ----------------------- | -------------------- - | ---------------- | | **In-Place Masking Mode** | Multi-Tenant | Available | | | Streams / Threads | Available | | | Batch Update | Available | | | Drop Indexes | Available | | | Disable Trigger | Available | | | Disable Constraint | Available | | | Identity Column Support | Available | | **On-The-Fly Masking Mode** | Restart Ability | Available | | | Truncate | Available | | | Disable Trigger | Available | | | Disable Constraint | Available | | **Profiling** | Multi-Tenant | Available | | | Streams | Available |

DB2 Z/OS and iSeries Connectors

Introduction

DB2 for z/OS and iSeries are relational database management systems that run on IBM Z(mainframe) and IBM Power Systems.

Support Matrix

| **i-Series** | | ---------- | | **7.1** | | **7.2** | | **7.3** | | **7.4** |

| **z/OS** | | ----------- | | **11** |

| | **Feature** | **Availability** | | ----------------------- | -------------------- | -------- -------- | | **In-Place Masking Mode** | Multi-Tenant | Available | | | Streams / Threads | Available | | | Batch Update | Available | | | Drop Indexes | Unavailable | | | Disable Trigger | Unavailable | | | Disable Constraint | Unavailable | | | Identity Column Support | Unavailable | | **On-The-Fly Masking Mode** | Restart Ability | Unavailable | | | Truncate | Available | | | Disable Trigger | Unavailable | | | Disable

Constraint | Unavailable | | **Profiling** | Multi-Tenant | Available | | | Streams | Available |

Files Connector

Introduction

Much of the time data will live outside of databases. The data can be stored in a variety of different formats including Fixed Width, Delimited, etc.

Support Matrix

File Type/Format	Support Level
Fixed Width	Supported
Delimited	Supported
XML	Supported
JSON	Not Supported

Mainframe Data Set Connector

Introduction

In addition to databases and files, the Masking Engine can process data stored in Mainframe data sets commonly found on the IBM z/OS operating system. For more information on data sets, see this IBM knowledge center article

Support Matrix

The Masking Engine requires that data be encoded in EBCDIC rather than something like ASCII or UTF-8. EBCDIC is the encoding traditionally used on Mainframes.

On-The-Fly Masking Jobs

Delphix Masking supports **On-The-Fly** (OTF) masking jobs where the data is read from a source location and written to a different target location. Only certain combinations of connector types are supported for OTF jobs.

OTF jobs with connectors of the same type are supported. For example, masking data from an Oracle source database to an Oracle target database is supported if both are using the built-in Oracle connector, as is a job with a delimited file source and target. OTF jobs using Extended Connectors are supported if both the source and target are using the same Extended Driver (the same uploaded JDBC driver). Additionally, OTF jobs with a relational database source and a delimited file target are supported. The following data sources are supported as source connectors for OTF jobs with delimited file targets.

- Oracle

- DB2

- MS SQL

- PostgreSQL

- MySQL / MariaDB

- SAP ASE (Sybase)

- Connectors created as Extended Connectors

No other combinations of connector types are supported. For example, an Oracle source with a PostgreSQL target, or an MS SQL source with a fixed width file target, are unsupported.

Installation

Prerequisites

This section will detail the hardware/software requirements needed to deploy the Delphix Engine with the Masking service. The Delphix Engine is a self-contained operating environment and application that is provided as a Virtual Appliance. Our Virtual Appliance is certified to run on a variety of platforms including VMware, AWS, and Azure.

The Delphix Engine should be placed on a server where it will not contend with other VMs for network, storage or other compute resources. The Delphix Engine is a CPU and I/O intensive application, and deploying it in an environment where it must share resources with other virtual machines can significantly reduce performance.

For those using both Delphix Virtualization and Delphix Masking, note that these must be deployed separately. A Delphix engine may only be used for either masking or virtualization. Running both masking or virtualization operation on one engine is not supported.

Client Web Browser

The Delphix Engine's graphical interface can be accessed from a variety of different web browsers. The Delphix Engine currently supports the following web browsers:

- Microsoft Edge 40.x or higher
- Mozilla Firefox 35.0 or higher
- Chrome 40 or higher

AWS EC2 Platform

See AWS EC2 Installation for information about the virtual machine requirements for installation of a dedicated Delphix Masking Engine on Amazon's Elastic Cloud Compute (EC2) platform.

Azure Platform

See Azure Installation for information about the virtual machine requirements for the installation of a dedicated Delphix Masking Engine on the Azure platform.

Google Cloud Platform

See Google Cloud Platform Installation for information about the virtual machine requirements for the installation of a dedicated Delphix Masking Engine on the GCP platform.

IBM Cloud

See IBM Cloud Installation for information about the virtual machine requirements for the installation of a dedicated Delphix Masking Engine on the IBM Cloud.

VMware Platform

See VMware Installation for information about the virtual machine requirements for the installation of a dedicated Delphix Masking Engine on the VMware Virtual platform.

AWS EC2 Installation

This section covers the virtual machine requirements for installation of a dedicated Delphix Masking Engine on Amazon's Elastic Cloud Compute (EC2) platform.

For best performance, the Delphix Masking Engine and all database/file servers should be in the same AWS region.

The following topics are covered:

- Instance Types

- Network Configuration

- EBS Configuration

- General Storage Configuration

- Additional AWS Configuration Notes

Instance Types

The Delphix Masking Engine can run on a variety of different instances, including large memory instances (preferred) and high I/O instances. We recommend the following large memory and high I/O instances:

Requirements	Notes
Large Memory Instances (**preferred**) r5n.2xlarge r5n.4xlarge r5n.8xlarge r5n.16xlarge r4.2xlarge r4.4xlarge r4.8xlarge r4.16xlarge	- Larger instance types provide more CPU, which can prevent resource shortfalls under high I/O throughput conditions. - Larger instances also provide more memory, which the Delphix Engine uses to cache database blocks. More memory will provide better read performance.
High I/O Instances (**supported**) i3.2xlarge i3.4xlarge i3.8xlarge	

Larger instance types provide more CPU, which can prevent resource shortfalls under high I/O throughput conditions.

> 🔥 **TIP - Estimating Delphix VM Memory Requirements**
>
> On the AWS EC2 platform, the Delphix Masking Engine must have sufficient memory to operate when multiple masking jobs are running. Our recommendation is to provide 8 GB of memory for the Delphix Masking Engine in addition to any memory that will be used by running jobs.

Network Configuration

Requirements	Notes
Virtual Private Cloud	- You must deploy the Delphix Engine and all of the source and target environments in a VPC network to ensure that private IP addresses are static and do not change when you restart instances. - When adding environments to the Delphix Engine, you must use the host's VPC (static private) IP addresses.
Static Public IP	The EC2 Delphix instance must be launched with a static IP address; however, the default behavior for VPC instances is to launch with a dynamic public IP address – which can change whenever you restart the instance. If you're using a public IP address for your Delphix Engine, static IP addresses can only be achieved by using assigned AWS Elastic IP Addresses.
Security Group Configuration	The default security group will only open port 22 for SSH access. You must modify the security group to allow access to all of the networking ports used by the Delphix Engine and the various source and target engines.

Storage Configurations

> ✏️ **Note**
>
> You must always attach a minimum of 2 storage pools to the Delphix Engine; one for rpool and other for domain0 pool.

EBS Configuration

Deploying Delphix on AWS EC2 requires EBS provisioned IOPS volumes. Since EBS volumes are connected to EC2 instances via the network, other network activity on the instance can affect throughput to EBS volumes. EBS optimized instances provide guaranteed throughput to EBS volumes and are required to provide consistent and predictable storage performance.

Requirements	Notes
EBS Provisioned IOPS Volumes **Note:** All attached storage devices must be EBS volumes.	- Delphix does not support the use of instance store volumes. - Use EBS volumes with provisioned IOPs in order to provide consistent and predictable performance. The number of provisioned IOPs depends on the estimated IO workload on the Delphix Engine. - Provisioned IOPs volumes must be configured with a volume size of at least 30 GiB times the number of provisioned IOPs. For example, for a provisioned volume of 3000 IOPs, you would need at least 100 GiB since the volume required is at least thirty times the provisioned IOP. - I/O requests of up to 256 kilobytes (KB) are counted as a single I/O operation (IOP) for provisioned IOPs volumes. Each volume can be configured for up to 4,000 IOPs.

System Disk

The minimum recommended storage size for the System Disk is 300 GB.

Metadata Disk(s)

The minimum recommended storage size of the Metadata Volume is 50 GB.

General Storage Configuration

Requirements	Notes
- Allocate initial storage equal to the size of the physical source database storage. - Add storage when storage capacity approaches 30% free.	- For high redo rates and/or high DB change rates, allocate an additional 10-20 %. - Add new storage by provisioning new volumes of the same size. - This enables the Delphix File System (DxFS) to make sure that its file systems are always consistent on disk without additional serialization. This also enables the Delphix Engine to achieve higher I/O rates by queueing more I/O operations to its storage.
EBS Volume Size and Count - Keep all EBS volumes the same size. Maximize Delphix Engine RAM for a larger system cache to service reads - Use at least 4 EBS volumes to maximize performance.	This enables the Delphix File System (DxFS) to make sure that its file systems are always consistent on disk without additional serialization. This also enables the Delphix Engine to achieve higher I/O rates by queueing more I/O operations to its storage.

Additional AWS Configuration Notes

- Using storage other than EBS is not supported.

- The use of the local SSDs attached to i2 instance types is not supported.

- Using fast storage for EBS volumes is supported and recommended, including (in order of decreasing speed):

  - Provisioned IOPS (io1) volumes (recommended).Virtual Machine Requirements for AWS EC2 Platform vLumen

  - General Purpose SSD (gp2) volumes (supported)

  - Throughput Optimized HDD (st1) volumes (supported)

  - Cold HDD (sc1) volumes (not supported due to poor performance)

  - Magnetic (standard) volumes (supported, but use st1 instead where possible)

Installing AMI on AWS EC2

The following two methods can be used to install/deploy Delphix Masking in AWS:

- Access Delphix provided AMI through the Delphix download site

- Subscribe to Delphix Masking through the Amazon Marketplace

Using the Delphix Download site to Deploy Masking

1. On the Delphix download site, click the AMI you would like to share and accept the Delphix License agreement. Alternatively, follow a link given by your Delphix solutions architect.

2. On the **Amazon Web Services Account** Details form presented:

   - Enter your **AWS Account Identifier**, which can be found here: https://console.aws.amazon.com/billing/home?#/account. If you want to use the GovCloud AWS Region, be sure to enter the ID for the AWS Account which has GovCloud enabled.

   - Select which **AWS Region** you would like the AMI to be shared in. If you would like the AMI shared in a different region, contact your Delphix account representative to make the proper arrangements.

3. Click **Share**. The Delphix Engine will appear in your list of AMIs in AWS momentarily.

4. Reference the Installation and Configuration Requirements for AWS/EC2 when deploying the AMI.

5. Once you have launched your Delphix Masking EC2 instance and it is accessible via a web browser (port 80), proceed to Setting up the Delphix Engine to configure the system.

Subscribing to Delphix Masking through Amazon Marketplace

1. Sign into the AWS Console.

2. Navigate to AWS Marketplace.

3. Typing Delphix in the search bar will find several Delphix Product offerings. Select **Delphix Masking for AWS (3TB)**.

4. Click **Continue to Subscribe**.

5. Click **Accept Terms**.

6. Wait for the subscription to be confirmed, then click **Continue to Configuration**.

7. Select or verify the correct **Region** for launch/deployment.

8. Then click **Continue to Launch**.

9. Select either to **Launch from Website** or **Launch through EC2**.

10. For either option you will need to enter the following:

    a. VPC in which to launch the instance.

    b. Subnet on which the instance will reside.

    c. Instance Type (Recommended: r4.2xlarge).

    d. Security Group (Minimal access required: 22, 80 or 443).

11. Once the Delphix EC2 instance is launched proceed to Setting up the Delphix Engine to configure the system.

Azure Installation

This topic covers the virtual machine requirements, including memory and data storage, for deploying the Delphix Engine on the Azure public cloud and Government Cloud.

Instance Types

The Delphix Engine can run on a variety of different Azure instances. We recommend the following instances:

Requirements	Notes
Memory-Optimizes	
DS14v2 E8S_v3 E16S_v3 E32S_v3	16 CPUs, 112GB, 32 devices 8 CPUs, 112GB, 16 devices 16 CPUs, 244GB, 32 devices 32 CPUs, 448GB, 64 devices Network bandwidth and IOPS limits are specific to each instance type: - See DSv2 specifications for more details. - See GS specifications for more details.
General Purpose	
D16s_v3 D32_v3	Network bandwidth and IOPS limits are specific to each instance type: - See DSv2 specifications for more details. - See DSv3 specifications for more details.

> 🔥 **TIP - Estimating Delphix VM Memory Requirements**
>
> On the Azure platform, commendation is to provide 8 GB of memory for the Delphix Masking Engine in addition to any memory that will be used by running jobs.

Network Configuration

Requirements	Notes
Azure Virtual Network (VNet)	The Delphix Engine and all the source and target environments must be accessible within the same virtual network.

Requirements	Notes
Network Security Group (NSG)	You must modify the security group to allow access to all of the networking ports used by the Delphix Engine and the various source and target platforms.

See Network Connectivity Requirements for information about specific port configurations.

Storage Configuration

> ✏️ **Note**
>
> You must always attach a minimum of 2 storage pools to the Delphix Engine; one for rpool and other for domain0 pool.

We recommend using a total of four disks to run your Delphix Engine. One disk is used for the Delphix File System (DxFS) to ensure that its file systems are always consistent on disk without additional serialization. The other three disks will be used for data storage. This also enables the Delphix Engine to achieve higher I/O rates by queueing more I/O operations to its storage.

Requirements	Notes
Azure Premium Storage	- Premium storage utilizes solid-state drives (SSDs) - Devices up to 4096GB are supported - Maximum of 256TB is supported - I/O requests of up to 256 kilobytes (KB) are counted as a single I/O operation (IOP) for provisioned IOPS volumes - IOPS vary based on storage size with a maximum of 7,500 IOPS
System Disk	The minimum recommended storage size for the System Disk is 300 GB.
Metadata Disk(s)	The minimum recommended storage size of the Metadata Volume is 50 GB.

Extensions

Extensions are not currently supported.

Installing VHD on AZURE

Use the following steps to install your VHD:

1. On the Microsoft Azure Marketplace, search for Delphix. Click **GET IT NOW**.

2. Reference the Installation and Configuration Requirements for the Delphix Engine in Azure when deploying the VHD.

3. Jump to Setting up the Delphix Engine section to learn how to activate the masking service now that you have the software installed.

Google Cloud Platform Installation

This section covers the virtual machine requirements for the installation of a dedicated Delphix Masking Engine on Google Cloud Platform (GCP).

Machine Types

The following is a list of instance types that are supported to deploy Delphix on GCP. Delphix periodically certifies new instance types, which will be added to the list here.

Requirements	Notes
n2-standard-(16, 32, 64)	Larger instance types provide more CPU, which can prevent resource shortfalls under high I/O throughput conditions.
n2-highmem-(8, 16, 32, 64)	Larger instances also provide more memory, which the Delphix Engine uses to cache database blocks. More memory will provide better read performance.

Network Configuration

Requirements	Notes
Virtual Private Cloud	You must deploy the Delphix Engine and database/file servers in a VPC network to ensure that private IP addresses are static and do not change when you restart instances. When adding connectors to the Masking Engine, you must use the host's VPC (static private) IP addresses.
Static Public IP	The GCP Delphix instance must be launched with a static IP address; however, the default behavior for VPC instances is to launch with a dynamic public IP address – which can change whenever you restart the instance.
Security Group Configuration	The default security group will only open port 22 for SSH access. You must modify the security group to allow access to all of the networking ports used by the Delphix Engine and the various source and target engines.
Premium Networking	It is recommended to use GCP Premium Tier Networking.

Storage Configuration

> ✏️ **Note**
>
> You must always attach a minimum of 2 storage pools to the Delphix Engine; one for rpool and other for domain0 pool.

SYSTEM DISK

The minimum recommended storage size for the System Disk is 300 GB.

METADATA DISK(S)

The minimum recommended storage size of the Metadata Volume is 50 GB.

Additional GCP Configuration Notes

- Delphix supports both Zonal and Regional SSD persistent disks.

Installing on Google Cloud Platform

This section covers the requirements, including memory and data storage, for deploying the Delphix Engine on Google Cloud Platform (GCP).

Prerequisites to Deploying in GCP

- A license is required to use the Delphix software. If you are a new customer contact Delphix to get started.

Deploying a Delphix Engine in GCP

1. Log into Google Cloud Marketplace with your account.

2. Search for **Delphix**.

3. Click **Launch on Compute Engine**.

    - Machine Type: See the table below for supported configurations.

    - Boot disk type: SSD Persistent Disk

    - Boot disk size in GB: 127

    - Networking interfaces: Configure as appropriate for your environment

    - IP forwarding: Configure as appropriate for your environment

4. Click on **Deploy**.

5. Once deployed, go to Setting up the Delphix Engine section to learn how to activate the masking service now that you have the software installed.

IBM Cloud Platform Installation

This topic covers the virtual machine requirements, including memory and data storage, for the deployment of the Delphix Engine on IBM Cloud.

Supported Profiles

The following is a list of profiles that are supported to deploy Delphix on IBM Cloud.

Requirements	Notes
mx2-8x64 mx2-16x128 mx2-32x256 mx2-48x384	- The Delphix Engine most closely resembles a storage appliance and performs best when provisioned using a storage-optimized profile - Larger profiles provide more CPU, which can prevent resource shortfalls under high I/O throughput conditions. - Larger profiles also provide more memory, which the Delphix Engine uses to cache database blocks. More memory will provide better read performance.

Network Configuration

Requirements	Notes
Virtual Server Instances	- You must deploy the Delphix Engine and all of the source and target environments in the same VPC network. - When adding environments to the Delphix Engine, you must use the host's VPC IP addresses.
Security Configuration	- The default security group will only open port 22 for SSH access. You must modify the security group to allow access to all of the networking ports used by the Delphix Engine and the various source and target engines. - See Network Performance Configuration Options for information about network performance tuning. - See General Network and Connectivity Requirements for information about specific port configurations. - Reference: IBM Cloud Security and Compliance documentation

Storage Configuration

> ✏️ **Note**
>
> You must always attach a minimum of 2 storage pools to the Delphix Engine; one for rpool and other for domain0 pool.

Requirements	Notes
- Allocate initial storage equal to the size of the physical source database storage. - Add storage when storage capacity approaches 30% free.	- For high redo rates and/or high DB change rates, allocate an additional 10-20 %. - Add new storage by provisioning new volumes of the same size. This enables the Delphix File System (DxFS) to make sure that its file systems are always consistent on disk without additional serialization. This also enables the Delphix Engine to achieve higher I/O rates by queueing more I/O operations to its storage. - A Delphix Engine requires a minimum of three (3) equally sized Block Volumes, in addition to the Boot volume which was automatically created while creating the virtual server instance. - IBM Block Storage Documentation

Additional IBM Configuration Notes

- Resize/expansion of a storage volume

- Expandable volume is a beta feature that is available for evaluation and testing purposes. This feature is available in the US South, US East, London, and France regions. Contact your IBM Sales representative if you are interested in getting access, [Expanding Block Storage[(https://cloud.ibm.com/docs/vpc?topic=vpc-expanding-block-storage-volumes/)

- After performing an "online" resize/expansion of a storage volume using IBM Cloud tools, then use the Delphix sysadmin interface to "Expand" the storage device; otherwise, the newly allocated storage space, from the resize/expansion, will not be used.

- Resize/expansion of a storage volume using IBM Cloud is not supported while the delphix engine is in a stopped state.

- Removing a storage volume

- It should be done while the machine is running.

- First use the Delphix sysadmin CLI interface to "Unconfigure" the storage device, then remove it from IBM Cloud.

Procedure for Deploying in the IBM Cloud

Prerequisites to Deploying in IBM Cloud

1. You require a license to use Delphix software. If you are a new customer, contact Delphix to get started.

2. Review IBM's cloud documentation for IBM Cloud specific information.

Deploying Delphix in the IBM Cloud

There are two methods for deploying a Delphix Engine in the IBM Cloud using the Software Catalog or Manually Uploading the Delphix Image.

Deploying from the IBM Software Catalog

1. Navigate to the IBM Software Catalog and search for Delphix.

2. Select the Delphix Data Masking Tile for the Masking product.

3. Scroll down to the Deployment Values section and input the specifics for your environment.

Required Parameters	Description
hostname	The name of the VSI you will use to deploy Delphix.
profile	Compute profile to be used for deploying Delphix (see recommended profiles).
ssh_key	Your public SSH key to be used when provisioning the VSI.
subnet_id	The id of the subnet where the VSI will be provisioned.
volumecount	Number of block storage volumes.
volumeprofile	Block storage profile to use (recommended is >= 10 IOPS/GB)
volumesize	Block storage volume size.
vpcname	The name of your VPC where the VSI is provisioned.
zone	VPC zone to provision your environment.

Manually Downloading and Deploying the Delphix Image

DOWNLOADING THE DELPHIX IMAGE

> ✏️ **Note**
>
> Contact your account manager to request access to the IBM variant of the Delphix product.

1. Follow the link given to you by your Delphix solutions architect. Download the Delphix_Verson…._Standard_IBM.qcow2 file and the SHA256SUMS file.

2. Once both files have finished downloading and assuming both files were downloaded to the same directory, you can run the following command to verify the download:

```
`$ grep -i IBM.qcow2 ./SHA256SUMS | sed -E 's,Appliance_Images/(Controlled_Availability/)?,,g' |
sha256sum --check`
```

Uploading the Delphix Engine Image as an Object

1. Authenticate with the IBM Cloud and navigate to the Dashboard.

2. Use the navigation menu to reach the **Resource List** page. The Resource List page can be navigated from the Dashboard by clicking on **Storage** within the **Resource summary** pane.

3. Expand Storage from the menu and select the appropriate resource group. You should have created a resource group depending on your organization's strategy for managing IBM resources.

4. Create a storage bucket or select an existing bucket.

5. Click the blue **Upload** button and select **Files**.

6. A pop-up menu appears to select the transfer type. **Aspera High-Speed Transfer** is required for large files. For this, you will need to install the plugin. It will automatically navigate you through the steps to install the plugin.

7. In the **Upload Files (objects)** window, click on the **Select Files (objects)** button and choose the IBM specific **QCOW2** file that was previously downloaded.

8. Click the **Upload** button.

Creating a Custom Image

1. Authenticate with IBM Cloud and navigate to the Dashboard.

2. Use the navigation menu to reach the **Custom images** page for VPC within the VPC infrastructure (IBM Cloud pull-down menu, upper left, VPC Infrastructure > Custom images).

3. Click the blue **Create** button.

4. In the **Import custom image** page, specify a unique name for the image.

5. From the **Resource Group** drop-down, select your organization's resource group.

6. Optional: In the **Tags** section, provide appropriate tags to organize your resources.

7. Select the appropriate **Region**.

8. Select the **Cloud Object Storage** bucket containing the uploaded image by selecting the appropriate **Cloud Object Storage instances > Location > Bucket** from the drop-down menus. The downloaded QCOW2 image should appear in the pane below the three drop-down menus.

9. Within the **Operating system** section, click on the **Ubuntu Linux** tile and select **ubuntu-18-04-amd64** from the drop-down menu.

10. Once all the parameters are entered, in the right pane click on the blue button to **Import custom image**.

Launching the Delphix Engine

1. Authenticate with IBM Cloud and navigate to the Dashboard.

2. Use the navigation menu to reach the **Virtual Server Instances** page within the VPC Infrastructure (IBM Cloud pull-down menu, upper left, VPC Infrastructure > Virtual Server Instances). Note: To maximize performance, deploy the Delphix Engine instance in the same VPC/subnet in which you will create your virtual databases (VDBs).

3. Click the blue **Create** button.

4. In the **New Virtual Server for VPC** page, specify a unique name for the VM.

5. From the **Virtual Private Cloud** drop-down, select your organization's VPC.

6. From the **Resource Group** drop-down, select your organization's resource group.

7. Optional: In the **Tags** section, provide appropriate tags to organize your resources.

8. Select the Location of your IBM Cloud resources.

9. In the **Operating System** section, click on the **Select Custom Image** link within the **Custom Image** block.

10. In the pop-menu, select the IBM specific image you previously uploaded.

11. Within the **Profile** section, click on **View all profiles**. Select one of the supported instance types and click Save.

12. You can skip the **User data** section.

13. You can also skip the **Boot Volume** section since it would already have the default values.

14. You can create block storage volumes later, so skip that for now. It will be discussed in the next section.

15. Continue on to the **Network Interfaces** section. If you already have a subnet configured in your zone and VPC, then this section will already have a default network interface. Otherwise, you need to create a subnet with the appropriate security groups. This part is critical, if the network isn't specified correctly, you are likely to run into firewall issues; please consult your IT or DevOps teams. Configure Network Security Groups (NSGs) for your subnet as required; again, please consult your IT or DevOps teams.

16. Click the **Create virtual server instance** button on the right panel. This will take a couple of minutes.

Creating Block Storage Volumes

1. Authenticate with IBM Cloud and navigate to the Dashboard.

2. Use the navigation menu to reach the **Block Storage Volumes** within VPC Infrastructure (IBM Cloud pull-down menu > VPC Infrastructure > Block Storage Volumes).

3. Click the blue **Create** button.

4. In the **Block Storage Volume for VPC** modal window, specify a unique name for this Block Volume. It can be helpful if this name is descriptive or identifies the VM it is intended to be attached to and ends in a sequence number.

5. From the **Resource Group** drop-down, select your organization's resource group.

6. Optional: In the **Tags** section, provide appropriate tags to organize your resources.

7. Select the Location of your IBM Cloud resources.

8. Enter the required IOPS. The recommended supported IOPS is 10/GB.

9. Enter the storage size in GB. Set the size of the volume to be sufficiently large, with room for growth, to support the databases that will be virtualized, or masked, by this Delphix Engine.

10. For **Encryption**, you can let it be the default, e.g. **Provider Managed**.

11. Click the blue **Create** Volume button. A Delphix Engine requires a minimum of three (3) equally sized Block Volumes, in addition to the Boot volume which was automatically created while creating the virtual server instance. Repeat Steps 3-11 as many times as necessary.

Attaching Block Storage Volumes

1. Authenticate with IBM Cloud and navigate to the Dashboard.

2. Use the navigation menu to reach the **Block Storage Volumes** within VPC Infrastructure (IBM Cloud pull-down menu > VPC Infrastructure > Block Storage Volumes).

3. From the list of pre-existing Block Volumes, identify the volumes you wish to attach to a Delphix Engine and wait until the volume's state becomes Available.

4. Note that the volumes you wish to attach have **Attachment Type** set as a **hyphen**.

5. The right side of the volume row shows an Expandable menu. Click on it and select **Attach to Instance**.

6. In the **Attach Virtual Server Instance** modal window, select your virtual server instance (Delphix Engine) from the drop-down menu.

7. Click on the blue **Attach Volume** button.

8. Repeat Steps 3-7 until all associated Block Volume resources have been attached to the Delphix Engine instance.

Configuring the Delphix Engine

1. Connect to your running Delphix Engine instance with a web browser. Use the IP address or DNS name noted in the Instance Description. Upon successful connection, the browser will display a login prompt to enter the Delphix Setup Page.

2. Refer to the standard product deployment instructions to complete your Delphix deployment.

Next Steps

Congratulations! You have successfully deployed a Delphix Engine in IBM Cloud.

Use Delphix documentation to learn how to:

- configure your database source

- configure your target environments

- create virtual databases (VDBs)

Hyper-V Installation

The Delphix Engine is a virtual appliance that runs in a hypervisor. In this section, you'll find requirements to run Delphix on Hyper-V including supported versions and instance configurations as well as recommended configuration parameters for optimal performance.

Contact your Delphix representative to request this capability. Delphix will assist you to review that all Hyper-V requirements are met to successfully run a Delphix Engine with the most appropriate configuration for your Use Cases.

If the Delphix Engine competes with other virtual machines on the same host for resources it will result in increased latency for all operations. As such, it is crucial that your Hyper-V host is not over-subscribed, as this eliminates the possibility of a lack of resources for the Delphix Engine. This includes allowing a percentage of CPU resources for the hypervisor itself as it can de-schedule an entire VM if the hypervisor is needed for managing IO or compute resources.

Supported Versions

- Hyper-V Version: 10.0 and later

- Gen 1 only is supported

Virtual CPUs

Requirements	Notes
8 vCPUs	- CPU resource shortfalls can occur both on an over-committed host as well as competition for host resources during high IO utilization. - CPU reservations are strongly recommended for the Delphix VM so that Delphix is guaranteed the full complement of vCPUs even when resources are overcommitted. - It is suggested to use a single core per socket unless there are specific requirements for other VMs on the same Hyper-V host.
Never allocate all available physical CPUs to virtual machines	- CPU for the Hyper-V Server to perform hypervisor activities must be set aside before assigning vCPUs to Delphix and other VMs. - We recommend that a minimum of 8-10% of the CPUs available are reserved for hypervisor operation. (e.g. 12 vCPUs on a 128 vCore system).

Memory

Requirements	Notes
128 or higher GB vRAM (recommended) 64GB vRAM (minimum)	- The masking service on the Delphix Engine uses its memory to process database and file blocks. - Memory reservations are required for the Delphix VM. The performance of the Delphix Engine will be significantly impacted by the over-commitment of memory resources in the Hyper-V Server. - Reservations ensure that the Delphix Engine will not be forced to swap pages during times of memory pressure on the host. A swapped page will require orders of magnitude more time to be brought back to physical memory from the Hyper-V swap device.
Memory for the Hyper-V Server to perform hypervisor activities must be set aside before assigning memory to Delphix and other VMs.	Failure to ensure sufficient memory for the host can result in a hard memory state for all VMs on the host which will result in a block for memory allocations.

Network

Requirements	Notes
Virtual ethernet adapter requirements.	- SR-IOV recommended for all virtual ethernet adapters that will be used for Delphix data IO. - Jumbo frames recommended. - A 10GbE NIC in the Hyper-V Server is recommended.
If the network load in the Hyper-V Server hosting the Delphix engine VM is high, dedicate one or more physical NICs to the Delphix Engine.	- Adding NICs only works if VMs are discovered using different interfaces.

SCSI Controller

Requirements	Notes
LSI Logic Parallel	- Per Hyper-V Storage I/O Performance Tuning Guidelines, it is recommended that you attach multiple disks to a single virtual SCSI controller and create additional controllers only as they are required to scale the number of disks connected to the virtual machine. For example, a VM with 3 virtual disks should distribute the disks across the single SCSI controller as follows: - IDE Controller 1 - Boot Drive - SCSI Controllers - Disk 1, Disk 2, Disk 3

Requirements	Notes
	Note: For load purposes, we generally focus on the DB storage and ignore the controller placement of the system disk.

Storage Configuration

> ✏️ **Note**
>
> You must always attach a minimum of 2 storage pools to the Delphix Engine; one for rpool and other for domain0 pool.

Requirements	Notes
Storage used for Delphix must be provisioned from storage that provides data protection.	For example, using RAID levels with data protection features, or equivalent technology. The Delphix Engine does not protect against data loss originating at the hypervisor or SAN layers.

Delphix Storage Operations

There are three types of data that Delphix stores on disk, which are:

1. Delphix VM Configuration Storage: stores data related to the configuration of the Delphix VM. VM Configuration Storage includes the Hyper-V configuration data as well as log files.

2. Delphix Engine System Disk Storage: stores data related to the Delphix Engine system data, such as the Delphix .ova settings.

Delphix VM Configuration Storage

The Delphix VM configuration must be stored on an NTFS volume(s).

Requirements	Notes
The volumes should have enough available space to hold all Hyper-V configuration and log files associated with the Delphix Engine.	If a memory reservation is not enabled for the Delphix Engine (in violation of memory requirements stated above), then space for a paging area equal to the Delphix Engine's VM memory must be added to the volumes containing the Delphix VM configuration data.

Delphix Engine System Disk Storage

Requirements	Notes
The Delphix Engine disks must be stored on NTFS volume(s).	The volume for the Delphix Engine System Disk Storage is often created on the same volume as the Delphix VM definition. In that case, the volume must have sufficient space to hold the Delphix VM Configuration, the virtual disk for the system disk, and a paging area if a memory reservation was not enabled for the Delphix Engine.
The Delphix .vhdx file is configured for a 128GB system drive.	The volume where the .vhdx is deployed should, therefore, have at least 128GB of free space prior to deploying the .vhdx.

Metadata Disk(s)

In addition to making sure the latest Hyper-V patches have been applied, check with your hardware vendor for updates specific to your hardware configuration. VHDXs (virtual machine disks).

Requirements	Notes
A minimum of 4 VHDXs should be allocated for database storage.	Allocating a minimum of 4 VHDXs for database storage enables the Delphix File System (DxFS) to make sure that its file systems are always consistent on disk without additional serialization. This also enables the Delphix Engine to achieve higher I/O rates by queueing more I/O operations to its storage.
If using VHDXs: - Each VHDX should be the only VHDX on its NTFS volume - The VHDX volumes should be assigned to dedicated physical LUNs on redundant storage. - The VHDXs should be created as the Fixed Size type.	Provisioning VHDXs from isolated volumes on dedicated physical LUNs: - Reduces contention for the underlying physical LUNs - Eliminates contention for locks on the volumes from other VMs and/or the Hyper-V Server Console
The quantity and size of VHDXs or RDMs assigned must be identical across all 4 controllers.	If the underlying storage array allocates physical LUNs by carving them from RAID groups, the LUNs should be allocated from different RAID groups. This eliminates contention for the underlying disks in the RAID groups as the Delphix engine distributes IO across its storage devices.
The physical LUNs used for NTFS volumes and RDMs should be of the same type in terms of performance characteristics such as latency, RPMs, and RAID level.	The total number of disk drives that comprise the set of physical LUNs should be capable of providing the desired aggregate I/O throughput (MB/sec) and IOPS (Input/Output Operations per Second) for all virtual databases that will be hosted by the Delphix Engine.

Requirements	Notes
The physical LUNs used for NTFS volumes can be thin-provisioned in the storage array.	If the storage array allocates physical LUNs from storage pools comprising dozens of disk drives, the LUNs should be distributed evenly across the available pools.

Installing Hyper-V

1. Download the image from Delphix's Download site and copy it to your VM directory.

2. Start the Hyper-V Manager and specify **Name and Location** and then select **Next**.

3. Specify the **Generation**, configure memory, and then select **Next**. Memory: 64 GB (minimum), 128 GB (recommended)

4. Set up Networking by selecting **vNIC** then select **Next**.

5. Attach the downloaded image as a boot disk. Create a unique boot disk for each image.
   Note: Boot disks cannot be shared.

   - Use an existing virtual hard disk.

   - Browse to the location of VM.

   - Select the Image.

6. Select **Finish**, the VM will appear in the inventory.

7. Customize the VM by selecting Settings:

   - Delphix recommends having the IDE be the first device to boot from (under BIOS setting).

   - Adjust the number of CPU (min 8)

   - Add Hard Drive. Use VHDX formatted disks. Recommend Fixed Size. **Note:** Differencing Disk Types are not supported.

   - 128 GB Disk Storage

8. Repeat step 7 as necessary.

9. Connect to the console and start the VM.

10. Once the installation is complete go to Setting up the Delphix Engine section to learn how to activate the masking service now that you have the software installed.

OCI Installation

This topic covers the virtual machine requirements for deploying the Delphix Masking Engine on Oracle Cloud Infrastructure (OCI).

Supported Databases

Oracle databases up to version 19c are supported. Please reference the Oracle Support Matrix for the detailed list.

Compute Image Types

Delphix distributes product images, for OCI, using the QCOW2 image type. Compute Images must be imported into OCI using the Paravirtualized launch mode; currently, images using the Emulated launch mode are not supported.

Supported Shapes

The following is a list of shapes that are supported to deploy Delphix on OCI.

Requirements	Notes
Large Memory Instances (perferred) VM.Standard2.8 VM.Standard2.16 VM.Standard2.24 VM.Standard.E4 Flex	The Delphix Engine most closely resembles a storage appliance and performs best when provisioned using a storage-optimized shape. Larger shapes provide more CPU, which can prevent resource shortfalls under high I/O throughput conditions. Larger shapes also provide more memory, which the Delphix Engine uses to cache database blocks. More memory will provide better read performance.

Network Configuration

Requirements	Notes
Virtual Cloud Network (VCN)	You must deploy the Delphix Engine and all of the source and target environments in a VCN to ensure that private IP addresses are static and do not change when you restart instances. By default, OCI subnets are considered public. When defining a subnet, we encourage configuring it as private. Unless required by your environment, your VCN should not include a Public Subnet. When adding environments to the Delphix Engine, you must use the host's VCN (static private) IP addresses.
Static Private IP	The Delphix instance should be launched with a static private IP address. For security reasons, it is encouraged to avoid configuring your engine with a Public IP address; but, in some cases,

Requirements	Notes
	it may be ok to use a dynamic Public IP address in addition to a static Private IP address if your environment requires such access.
Security Rules Configuration	OCI allows two firewall features: Network Security Groups (NSGs) and Security Lists. Oracle recommends the use of NSGs over Security Lists because "NSGs let you separate the VCN's subnet architecture from your application security requirements." However, a VCN will use a Security List to define default rules. By default, the security list will only open port 22 for SSH access. You must modify the security list, or create NSGs, to allow access to all of the networking ports used by the Delphix Engine and the various source and target engines. This dual implementation of firewall, or security, rules may be different from other clouds. Please see OCI documentation for best practices. See Network Connectivity Requirements for information about specific port configurations.

Storage Configuration

> ✏️ **Note**
>
> You must always attach a minimum of 2 storage pools to the Delphix Engine; one for rpool and other for domain0 pool.

Requirements	Notes
Allocate initial storage equal to the size of the physical source database storage. Attach a minimum of four (4), equally sized, storage devices to the Delphix Engine. Add storage when storage capacity approaches 30% free.	Currently supported Instance Types, or Shapes, only support Block Volumes; File Storage is not supported. Paravirtualized block devices are required; currently, iSCSI devices are not supported. Elastic Performance Configuration Options (aka Volume Performance Policy): use Higher Performance. For high redo rates and/or high DB change rates, allocate an additional 10-20 %.
Must use Block Volume for data storage. Block Volumes must be attached using Paravirtualized mode.	Add new storage by provisioning new volumes of the same size. This enables the Delphix Engine to achieve higher I/O rates by distributing load among devices and queueing more I/O operations to its storage.

Additional OCI Configuration Notes

- When running low on storage space, Delphix recommends adding additional equivalently sized block storage volumes, or devices, instead of resizing existing volumes.

- If you must expand existing storage volumes, then this must be done using the "online" resizing strategy specified in OCI documentation; "offline" storage resizing is not supported and may lead to unexpected downtime. If an existing storage volume is expanded, then use the Setup, or sysadmin, interface to expand each storage "device," or volume. The additional storage, as a result of a resize, will not be available for use until the storage devices are explicitly instructed to make use of the additional space.

- If expanding storage volumes, it is recommended that all volumes are expanded to the same size. When storage volumes, or devices, are the same size the Delphix product is able to balance I/O distribution among the disks for optimal performance.

- Hot removal of storage volumes is not supported.

Installing OCI

Download and Verify the Delphix Engine Image

1. Contact your account manager to request access to the OCI variant of the Delphix product.

2. Follow the link given by your Delphix solutions architect. Download the `Delphix_6.x.x.x_….Standard_OCI.qcow2` file and the `SHA256SUMS` file.

3. Once both files have finished downloading and assuming both files were downloaded to the same directory, you can run the following command to verify the download: `$ grep -i OCI.qcow2 ./SHA256SUMS | sed -E 's,Appliance_Images/(Controlled_Availability/)?,,g' | sha256sum --check`

Upload the Delphix Engine Image as an Object

1. Authenticate with OCI and navigate to the [Infrastructure Console](#).

2. Use the navigation menu to reach the **Object Storage Buckets, Core Infrastructure**, page (Hamburger Menu > Object Storage > Object Storage).

3. Remember to set your **List Scope Compartment**. This will depend on your organization's strategy for managing OCI resources.

4. [Create a storage bucket](#) or select an existing bucket.

5. Click the blue **Upload** button.

6. In the **Upload Objects** modal window, specify an optional prefix and choose the OCI specific QCOW2 file that was previously downloaded.

7. Click the blue **Upload** button.

Creating a Custom Compute Image from an Object

1. Authenticate with OCI and navigate to the **Infrastructure Console**.

2. Use the navigation menu to reach the **Compute Custom Images, Core Infrastructure**, page (Hamburger Menu > Compute > Custom Images).

3. Remember to set your **List Scope Compartment**. This will depend on your organization's strategy for managing OCI resources.

4. Click the blue **Image Import** button.

5. In the **Import Image** modal window, select a suitable compartment in the **Create In Compartment** field that conforms to your organization's strategy on managing OCI resources.

6. In the **Name** field enter a unique name to identify the Custom Compute Image. You may want to use the same, resulting name of the image object from the previous step, Upload the Delphix Engine Image as an Object.

7. For **Operating System** select **Linux**.

8. Next, identify an object by specifying its Compartment, Bucket, and Object Name. Or, specify an Object Storage URL. **Note:** The Object Details will identify this value as **URL Path (URI)**.

9. For **Image Type** select **QCOW2**.

10. For **Launch Mode** select **Paravirtualized Mode**.

11. For organizations that have a tagging policy for cloud-based resources, expand the **Tagging Options** section, and define tags.

12. Click the blue **Import Image** button.

Launching the Delphix Engine

1. Authenticate with OCI and navigate to the **Infrastructure Console**.

2. Use the navigation menu to reach the **Compute Instances, Core Infrastructure**, page (Hamburger Menu > Compute > Instances).

3. Remember to set your **List Scope Compartment**. This will depend on your organization's strategy for managing OCI resources.

4. Click the blue **Create Instance** button.

5. In the **Create Compute Instance** window pane, specify a unique name for the VM.

6. For the **Create In Compartment** field, select a suitable compartment that conforms to your organization's strategy on managing OCI resources.

7. In the **Image or operating system** section, click the Change Image button. Switch to the Custom Images tab. Find the Delphix image that corresponds to the instance you wish to deploy. Click the blue **Select Image** button. **Note:** If the Delphix Custom Image is not visible, look for the **Change Compartment** option near the top of the current window pane.

8. Each Availability Domain has its own quota, it is ok to use AD-1, AD-2, or AD-3 - but, be sure to make note of which Availability Domain you are using. **Note:** Compute Instances and attached Storage will need to be in the same Availability Domain.

9. In the **Shape** section click the **Change Shape** button. For **Instance type** specify **Virtual Machine** and for **Shape series** use **Intel Skylake**. Then select an OCI Shape that is supported by Delphix.

10. Continue on to the **Configure networking** section. This part is critical, if the network isn't specified correctly, you are likely to run into firewall issues; please consult your IT or DevOps teams. If your organization is using Network Security Groups (NSGs), mark the **Use Network Security Groups to Control Traffic** checkbox; again, please

consult your IT or DevOps teams. Lastly, select the Do Not Assign a Public IP Address radio button; if you must deviate from this guidance then you are highly encouraged to engage your IT or DevOps teams.

11. You may skip the **Boot Volume** section.

12. In the **Add SSH Keys** select the **No SSH Keys** radio option. The Delphix product is a closed appliance and manages users independently.

13. In general, you can skip all of the Advanced Options. For organizations that have a tagging policy for cloud-based resources, expand into the Advanced Management section, and look for the Tagging sub-section to define tags.

14. Click the blue **Create button** - wait about 2-5 minutes for the Delphix Engine instance to boot.

Create Block Storage Volumes

1. Authenticate with OCI and navigate to the **Infrastructure Console**.

2. Use the navigation menu to reach the **Block Volumes, Core Infrastructure**, page (Hamburger Menu > Block Storage > Block Volumes).

3. Remember to set your **List Scope Compartment**. This will depend on your organization's strategy for managing OCI resources.

4. Click the blue **Create Block Volume** button.

5. In the **Create Block Volume**modal window, specify a unique name for this Block Volume. It can be helpful if this name is descriptive or identifies the VM it is intended to be attached to and ends in a sequence number.

6. For the **Availability Domain**, this value MUST be the same Availability Domain used for the Delphix Engine instance, otherwise, this volume will not be available for use.

7. In the **Volume Size and Performance** section, select the **Custom** option. Set the size of the volume to be sufficiently large, with room for growth, to support the databases that will be virtualized, or masked, by this Delphix Engine. And, set the **Default Volume Performance** to the **Higher Performance** setting.

8. A **Backup Policy** is not required and can be left blank or **No Backup Policy Selected**. However, depending on your organization's needs, you may consider selecting a Backup Policy.

9. For **Encryption**, it is ok to use the default option, **Encrypt Using Oracle-Managed Keys**. Optionally, if you want, or need, to manage encryption keys independently then use the Encrypt Using Customer-Managed Keys option.

10. For organizations that have a tagging policy for cloud-based resources, expand the **Tagging Options** section, and define tags.

11. Uncheck the checkbox that says **View Detail Page After This Block Volume is Created**. This will prevent you from navigating away from the Block Volumes page, because, more often than not, you will need to create multiple Block Volumes at the same time.

12. Click the blue **Create Block Volume** button.

13. A Delphix Engine requires a minimum of four (4) equally sized Block Volumes. Repeat Steps 4-12 as many times as necessary.

Attach Block Storage Volumes

1. Authenticate with OCI and navigate to the **Infrastructure Console**.

2. Use the navigation menu to reach the **Block Volumes, Core Infrastructure**, page (Hamburger Menu > Block Storage > Block Volumes).

3. Remember to set your **List Scope Compartment**. This will depend on your organization's strategy for managing OCI resources.

4. From the list of pre-existing Block Volumes, identify the resources you wish to attach to a Delphix Engine and wait until the volume's state becomes Available.

5. Select one of the **Block Volumes** to enter the **Block Volume Details** page.

6. On the left-hand side, locate the **Resources** menu and select **Attached Instances**.

7. If the Block Volume has not been previously attached to another VM, then you will be able to click the blue **Attach to Instance** button.

8. In the Attach to Instance modal window, specify the **Attachment Type** as **Paravirtualized**. Currently, iSCSI is not supported.

9. For **Access Type** use the **READ/WRITE** option.

10. Next, identify a Delphix Engine by selecting an instance, or by specifying an instance OCID. If you don't see the Delphix Engine instance in the Select an Instance drop-down menu, you may need to use the Change Compartment option. Block Volumes can only be attached to VM instances that were created in the same Availability Domain - if these values do not match, you will need to either re-provision Block Volumes or the Delphix Engine, in the correct Availability Domain.

11. Click the blue Attach button.

12. Repeat Steps 4-11 until all associated Block Volume resources have been attached to the Delphix Engine instance.

Configuring Masking

Once deployed, go to First Time Setup section to learn how to activate the masking service now that you have the software installed.

VMware Installation

The Delphix Engine is a virtual appliance that runs in a hypervisor. In this section, you'll find requirements to run Delphix on VMware including supported versions and instance configurations as well as recommended configuration parameters for optimal performance.

The Delphix Engine is intensive both from a network and a storage perspective. If the Delphix Engine competes with other virtual machines on the same host for resources it will result in increased latency for all operations. As such, it is crucial that your ESXi host is not over-subscribed, as this eliminates the possibility of a lack of resources for the Delphix Engine. This includes allowing a percentage of CPU resources for the hypervisor itself as it can de-schedule an entire VM if the hypervisor is needed for managing IO or compute resources.

Supported ESX Versions

Requirements	Notes
VMware Cloud VMware ESX/ESXi 7.0, 7.0 U1 ESX/ESXi 6.7 U3 VMware ESX/ESXi 6.5 U1, 6.5 U3 VMware ESX/ESXi 6.0	More recent versions of VMware are preferred, such as ESX/ESXi 6.0 - 7.0

Virtual CPUs

Requirements	Notes
8 vCPUs	- CPU resource shortfalls can occur both on an over-committed host as well as competition for host resources during high IO utilization. - CPU reservations are strongly recommended for the Delphix VM so that Delphix is guaranteed the full complement of vCPUs even when resources are overcommitted. - It is suggested to use a single core per socket unless there are specific requirements for other VMs on the same ESXi host.
Never allocate all available physical CPUs to virtual machines	- CPU for the ESXi Server to perform hypervisor activities must be set aside before assigning vCPUs to Delphix and other VMs. - We recommend that a minimum of 8-10% of the CPUs available are reserved for hypervisor operation. (e.g. 12 vCPUs on a 128 vCore system).

Memory

Requirements	Notes
128 or higher GB vRAM (recommended) 64GB vRAM (minimum)	- The masking service on the Delphix Engine uses its memory to process database and file blocks. - More memory can sometimes improve performance. Memory reservation is a requirement for the Delphix VM. - Overcommitting memory resources in the ESX server will significantly impact the performance of the Delphix Engine. - Reservation ensures that the Delphix Engine will not stall while waiting for the ESX server to page in the engine's memory.

> 🔥 **TIP - Do not allocate all memory to the Delphix VM**

Never allocate all available physical memory to the Delphix VM. You must set aside memory for the ESX Server to perform hypervisor activities before you assign memory to Delphix and other VMs. The default ESX minimum free memory requirement is 6% of the total RAM. When free memory falls below 6%, ESX starts swapping out the Delphix guest OS. We recommend leaving about 8-10% free to avoid swapping

For example, when running on an ESX Host with 512GB of physical memory, allocate no more than 470GB (92%) to the Delphix VM (and all other VMs on that host).

Network

Requirements	Notes
The ova is pre-configured to use one virtual ethernet adapter of type VMXNET 3.	- Jumbo frames are highly recommended to reduce CPU utilization, decrease latency, and increase network throughput. (typically 10-20% throughput improvement) - If additional virtual network adapters are desired, they should also be of type VMXNET 3.
A 10GbE NIC in the ESX Server is recommended.	For VMs having only gigabit networks, it is possible to aggregate several physical 1GbE NICs together to increase network bandwidth (but not necessarily to reduce latency). Refer to the VMware Knowledge Base article NIC Teaming in ESXi and ESX. However, it is not recommended to aggregate NICs in the Delphix Engine VM.

Storage

Overview

> ✏️ **Note**
>
> You must always attach a minimum of 2 storage pools to the Delphix Engine; one for rpool and other for domain0 pool.

There are three types of data that Delphix stores on disk, which are:

1. **Delphix VM Configuration Storage:** stores data related to the configuration of the Delphix VM. VM Configuration Storage includes the VMware ESX configuration data as well as log files.

2. **Delphix Engine System Disk Storage:** stores data related to the Delphix Engine system data, such as the Delphix .ova settings.

3. **Metadata Storage:** stores metadata used by the Masking service.

General Requirements

Requirements	Notes
Storage used for Delphix must be provisioned from storage that provides data protection.	For example, using RAID levels with data protection features, or equivalent technology. The Delphix engine product does not protect against data loss originating at the hypervisor or SAN layers.

Delphix VM Configuration Storage

The Delphix VM configuration should be stored in a VMFS volume (often called a "datastore").

Requirements	Notes
The VMFS volume should have enough available space to hold all ESX configuration and log files associated with the Delphix Engine.	If a memory reservation is not enabled for the Delphix Engine (in violation of memory requirements stated above), then space for a paging area equal to the Delphix Engine's VM memory must be added to the VMFS volume containing the Delphix VM configuration data.

Delphix Engine System Disk Storage

The VMFS volume must be located on shared storage in order to use vMotion and HA features.

Requirements	Notes
The Delphix Engine system disk should be stored in a VMDK.	The VMDK for the Delphix Engine System Disk Storage is often created in the same VMFS volume as the Delphix VM definition. In that case, the datastore must have

Requirements	Notes
	sufficient space to hold the Delphix VM Configuration, the VDMK for the system disk, and a paging area if a memory reservation was not enabled for the Delphix Engine.
The Delphix .ova file is configured for a 127GB system drive.	The VMFS volume where the .ova is deployed should, therefore, have at least 127GB of free space prior to deploying the .ova.

Delphix Engine Metadata Storage

Shared storage is required in order to use vMotion and HA features. In addition to making sure the latest VMware patches have been applied, check with your hardware vendor for updates specific to your hardware configuration. VMDKs (Virtual Machine Disks) or RDMs (Raw Device Mappings) operating in virtual compatibility mode can be used for data storage.

Requirements	Notes
The minimum recommended storage size is 50 GB.	

In addition to making sure the latest VMware patches have been applied, check with your hardware vendor for updates specific to your hardware configuration.

Additional VMware Configuration Notes

- Running Delphix inside of vSphere is supported.
- Using vMotion on a Delphix VM is supported.
- Device passthrough is not supported.

Installing OVA on VMware

1. Download the OVA file from Delphix's Download site. Note, you will need a support login from your sales team or welcome letter. Navigate to "Virtual Appliance" and download the appropriate OVA. If unsure, use the HWv11 OVA type.

2. Login using the vSphere client to the vSphere server (or vCenter Server) where you want to install the Delphix Engine.

3. In the vSphere Client, click **File**.

4. Select **Deploy OVA Template** and then browse to the OVA file. Click **Next**.

5. Select a hostname for the Delphix Engine. This hostname will be used in configuring the Delphix Engine network.

6. Select the data center where the Delphix Engine will be located.

7. Select the cluster and the ESX host.

8. Select one (1) data store for the Delphix OS. This datastore can be thin-provisioned and must have enough free space to accommodate the 300GB comprising the Delphix operating system.

9. The Delphix VM Configuration Storage requires a minimum of 10GB. The VMFS volume should have enough available space to hold all ESX configuration and log files associated with the Delphix Engine.

   The Delphix Engine system disk should be stored in a VMDK system drive. The VMFS volume where the .ova is deployed should therefore have at least 300GB of free space prior to deploying the .ova. The VMFS volume must be located on shared storage in order to use vMotion and HA features.

10. Select the virtual network you want to use. If using multiple physical NICs for link aggregation, you must use vSphere NIC teaming. Do not add multiple virtual NICs to the Delphix Engine itself. The Delphix Engine should use a single virtual network.

11. Click **Finish**. The installation will begin and the Delphix Engine will be created in the location you specified.

12. Once the Delphix Engien has been created proceed to Setting up the Delphix Engine to configure the system.

Network Connectivity Requirements

This topic covers the general network and connectivity requirements, including connection requirements, port allocation, and firewall and Intrusion Detection System (IDS) considerations.

General Outbound Connections from the Delphix Masking Engine

Protocol	Port Numbers	Use
TCP	25	Connection to a local SMTP server for sending email.
TCP/UDP	53	Connections to local DNS servers.
UDP	123	Connection to an NTP server.
UDP	162	Sending SNMP TRAP messages to an SNMP Manager.
TCP	443	HTTPS connections from the Delphix Engine to the Delphix Support upload server.
TCP/UDP	636	Secure connections to an LDAP server.
TCP/UDP	various	Connections to target environments such as databases (JDBC) and files (FTP, SFTP, NFS, or CIFS).

General Inbound Connections to the Delphix Masking Engine

Protocol	Port Numbers	Use
TCP	22	SSH connections to the Delphix Engine.
TCP	80	HTTP connections to the Delphix GUI (optional).
UDP	161	Messages from an SNMP Manager to the Delphix Engine.
TCP	443	HTTPS connections to the Delphix GUI.

Firewalls and Intrusion Detection Systems (IDS)

Firewalls can add milliseconds to the latency between servers. Accordingly, for best performance, there should be no firewalls between the Delphix Masking Engine and the target environments. If the Delphix Masking Engine is separated from a target environment by a firewall, the firewall must be configured to permit network connections between the Delphix Masking Engine and the target environments for the application protocols (ports) listed above.

Intrusion detection systems (IDSs) should also be made permissive to the Delphix Masking Engine deployment. IDSs should be made aware of the anticipated high volumes of data transfer between the Delphix Masking Engine and target environments.

First Time Setup

This section walks you step by step on how to download and install the Delphix Engine software onto your infrastructure (VMware, AWS EC2, Azure, or GCP).

Setting Up Network Access to the Delphix Engine

1. Power on the Delphix Engine and open the Console.

2. Wait for the Delphix Management Service and Delphix Boot Service to come online.   This might take up to 10 minutes during the first boot. Wait for the large orange box to turn green.

3. Press any key to access the sysadmin console.

4. Enter **sysadmin** for the username and **sysadmin** for the password (when installing a new engine via AWS AMI, the initial sysadmin password is the AWS Instance ID).

5. You will be presented with a description of available network settings and instructions for editing.

6. Configure the hostname. Use the same hostname you entered during the server installation. If you are using DHCP, this step can be skipped.

7. Configure DNS. If you are using DHCP, this step can be skipped.

8. Configure either a static or DHCP address. The static IP address must be specified in CIDR notation (for example, 192.168.1.2/24).

9. Configure a default gateway. If you are using DHCP, this step can be skipped.

10. Commit your changes. Note that you can use the get command prior to committing to verify your desired configuration.

11. Check that the Delphix Engine can now be accessed through a Web browser by navigating to the displayed IP address, or hostname if using DNS.

12. Exit setup.

Setting up the Delphix Engine

Once you setup the network access for your Delphix Engine, enter the Delphix Engine URL in your browser for server setup. The Unified Setup wizard Welcome screen below will appear for you to begin your Delphix Engine setup.

The Welcome page allows you to setup Masking-specific settings such as Masking admin user's email and password as well as Masking SMTP settings directly from the setup wizard. It will then redirect the customer to the corresponding login page based on the engine type selected.

When Masking is selected, the following will be added to the Welcome screen; "admin" with the password you defined. This will be the Masking administrator responsible for setting up users and other administrative actions in Masking.

There are limitations to this feature:

- Only Masking user settings (email and password) and SMTP settings are supported. Customers will need to use the API to setup LDAP.

- Once set, these settings can only be updated via the Masking API. There are no corresponding sections in the system dashboard.

- Engine Type cannot be modified once set in the Setup Wizard because it has other dependencies such as SSO.

> ✎ **Note**
>
> If the wrong password is entered, after 3 times the user will be locked out of the Masking service.

1. On the **Welcome** tab select **Masking** and then click **Next**.

2. In the Masking Password tab enter the current default (out-of-box) password for Masking. (Currently, the default is **Admin-12**)

3. Click **Validate** or **Next**. This causes the engine to validate the entered password with the masking service.

4. In the Administrators tab enter **System Administrator**, **Masking Administrator**, and **Engine Administrator** credentials. Then click **Next**.

5. Select an option for maintaining system time. Then click **Next**.

6. Configure your network interfaces and services and then select **Next**.

7. Delphix installs certificates signed by the Engines Certificate Authority. You can replace any certificate. Once you are ready click **Next**.

8. The Delphix Engine automatically discovers and displays storage devices. For each device, set the Usage Assignment to Data and set the Storage Profile to Striped. Then click **Next**.

9. Enter the **Masking SMTP** settings and then click **Next**.

10. The Authentication tab allows users to configure Virtualization LDAP settings. But Masking LDAP settings must be configured via the Masking API.

11. To enable SAML/SSO, set the Audience Restriction (SP entity ID, Partner's Entity ID) in the identity provider to be the Engine UUID. Select **Use SAML/SSO**.IdP metadata is an XML document which must be exported from the application created in your IdPCopy and pasted in the IdP Metadata field. Click **Next**.

12. If using Kerberos authentication select **Use Kerberos authentication** and complete all fields. Then enter **Next**.

13. If the Delphix Engine has access to the external Internet (either directly or through a web proxy), then you can auto-register the Delphix Engine. If external connectivity is not immediately available, you must perform manual registration. Copy the Delphix Engine registration code.

14. Click **Next**.

15. The final Summary tab will enable you to review your configuration. Click **Submit** to acknowledge the configuration.

Logging in to the Delphix Masking Engine

1. Login to a web browser that points to http://masking-engine.example.com/masking.

2. Enter default username: admin.

3. Enter default user password: Admin-12

Naming Requirements

This section describes the naming requirements for Masking Engine objects which are allowed to be created / renamed manually.

Affected configurable objects

configurable objects
algorithm
application
connector
domain
environment
file format
job
profiling group
record type
role
rule set
search expression

For all of the above:

- Leading/trailing white space is not allowed

- The following special characters are not allowed:

Symbol	Name
[	open bracket
]	close bracket
(	open parenthesis
)	close parenthesis
{	open brace
}	close brace
~	tilde
!	exclamation mark
@	at
#	pound
$	dollar
%	percent
^	carat
*	asterisk
"	quote
?	question mark
:	colon
;	semi-colon
,	comma
/	forward slash

Symbol	Name
\	back slash
\\	double back slash
`	back quote
+	plus
=	equal
<	less than
>	greater than
'	single quote
|	pipe

Upgrade

During an upgrade of a Masking Engine to a 6.0 or later release, a name with leading or trailing white space will be automatically trimmed, and a counter value might be appended to the end of the name to prevent a naming conflict. For example:

pre-upgrade name	post-upgrade name	upgrade change
"alg_SecureLookup"	"alg_SecureLookup"	no change
" alg_SecureLookup"	"alg_SecureLookup1"	leading white space trimmed and counter value appended
"alg_SecureLookup "	"alg_SecureLookup2"	trailing white space trimmed and counter value appended

If any name from the above mentioned "configurable entities" table has a restricted special character - an upgrade will fail with the corresponding error message.

Create / Rename

If an attempt is made to create a new entity (or to modify the name of the existing one) with leading or trailing white space or any of the special characters listed above, the operation will fail on a 6.0 or later release with a corresponding error message.

Environment Export - Import

If any entity name exported from a pre-6.0 version contains leading or trailing white spaces or the special characters listed above, the import operation will fail on a 6.0 or later release with a corresponding error message.

Sync

If a sync bundle from a pre-6.0 version contains leading or trailing white space or any of the special characters listed above, then the Sync import operation will fail on a 6.0 or later release, with a corresponding error message.

Users and Roles

The Delphix Masking Service has a flexible and robust users and roles system that allows you to give users fine-grain privileges over what environments they have access to and what tasks they can and can not perform.

What are Roles?

A defined role is what is used to give a certain user privileges over certain environments and tasks. Roles can be defined by selecting a subset of actions that can be taken on certain objects.

Actions

When defining a role, you can select one or more of the following actions for the role to be able to perform:

- **View:** Be able to view the object and important information about the object.
- **Add:** Be able to add an instance of an object.
- **Update:** Be able to update/edit an instance of an object.
- **Delete:** Be able to delete an instance of an object.
- **Copy:** Be able to create a copy of an object.
- **Export:** Be able to export an object from a Delphix Engine.
- **Import:** Be able to import an exported object into a Delphix Engine

Please note that not all of these actions are available for all objects in the masking service.

Objects

When defining a role, permission to perform the above actions can be defined on a per-object basis. These objects include:

General	Jobs	Settings
Environment	Profile Job	Domains
Connection	Masking Job	Algorithms
Ruleset		Profiler
Inventory		Profiler Set
		Custom Algorithms

General	Jobs	Settings
		File Format
		Users
		Diagnostic

Refer to Delphix Masking Terminology for definitions of these objects.

Adding A Role

To add a role follow these steps:

1. Login into the **Masking Engine** and select the **Settings** tab.

2. Click the **Add Roles** button.

3. Enter a **Role Name**. The far-left column lists the items for which you can set privileges.

4. Select the checkboxes for the corresponding privileges that you want to apply. If there is no checkbox, that privilege is not available. For example, if you want this role to have View, Add, Update, and Run privileges for masking jobs, select the corresponding checkboxes in the **Masking Job** row.

5. When you are finished assigning privileges for this Role, click **Submit**.

Recommended Roles

While every organization will differ in what users and roles they define, we have found that the following roles are common/popular:

- **Administrator** — Unrestricted access to all the engine functions. The Admin role is assigned via the checkbox in the add user page of the UI.
- **IT Security Analyst** — Unrestricted access for all settings functions; access to all application functions except environment and environment create, delete, update.

> ✏️ **IT Security Analyst role JSON**   ⌄
>
> Sample JSON

- **All Privileges** — Unrestricted access for an application environment; central admin or security analyst will determine if this role can modify settings.

✏️ **All Privileges role JSON**   ⌄

Sample JSON

- **DBA** — Manage connections for application database, scripting and scheduling (no settings).

  ✏️ **DBA role JSON**   ⌄

  Sample JSON

- **SME/Analyst/Developer** — Manage inventories, create, view jobs.

  ✏️ **SME/Analyst/Developer role JSON**   ⌄

  Sample JSON

- **Operator** — All job privileges.

  ✏️ **Operator role JSON**   ⌄

  Sample JSON

- **Environment Owner** — Approve workflow and inventories, privileges to view for settings and environment.

  ✏️ **Environment Owner role JSON**   ⌄

  Sample JSON

✏️ **NOTE - One Role per User**

Please note that each defined user can only have one role assigned to them.

What are Users?

Once you have your roles defined, it is time to create users with those roles. We highly recommend creating independent users for each individual who will have access to the masking service.

Adding a User

To create a new user using the Masking UI follow these steps:

1. Login into the **Masking Engine** and select the **Admin** tab.

2. Click **Add User** at the upper right of the Users screen.

3. You will be prompted for the following information:

- **First Name** — (Optional) The user's given name

- **Last Name** — (Optional) The user's surname

- **User Name** — The login name for the user

- **Email** — The user's e-mail address (mailable from the Delphix Masking Engine server for purposes of job completion e-mail messages)

- **Password** — The password that the Delphix Masking Engine uses to authenticate the user on the login page. The password must be at least six characters long but no longer than 12 characters, and contain a minimum of one uppercase character, one wild character (!@#$%^&*), and one number.

- **Confirm Password** — Confirm the password with double-entry to avoid data entry errors.

- **Administrator** — (Optional) Select the Administrator checkbox if you want to give this user Administrator privileges. (Administrator privileges allow the user to perform all Delphix Masking Engine tasks, including creating and editing users in the Delphix Masking Engine.) If you select the Administrator checkbox, the Roles and Environments fields disappear because Administrator privileges include all roles and environments.

- **Role** — Select the role to grant to this user. The choices here depend on the custom roles that you have created. You can assign one role per user name.

- **Environment** — Enter as many environments as this user will be able to access. Granting a user access to a given environment does not give them unlimited access to that environment. The user's access is still limited to their assigned role.

4. When you are finished, click **Save**.

> ✏️ **Note**
>
> When a user is created, it's Account Status is *Active* by default.

To create a new user using the Masking API follow these steps:

1. Access the API client on your Masking Engine, from http://myMaskingEngine.myDomain.com/masking/api-client.

2. Login into the **Masking Engine** and select the **User** endpoint.

user		Show/Hide | List Operations | Expand Operations
GET	/users	Get all users
POST	/users	Create user
DELETE	/users/{userId}	Delete user by ID
GET	/users/{userId}	Get user by ID
PUT	/users/{userId}	Update user by ID
POST	/users/forgot-password	Send Reset password mail to the user
POST	/users/reset-password	Reset new password for the user

3. Click **Create users** at the upper right of **/users** section and refer to the **Example Value** for parameters required for new users.

4. Enter valid User creation JSON in the **body** section, refer to sample create users JSON.

   Sample New User Create JSON

5. Click on **Execute API Request**.

Updating A User

To update a user information using Masking UI follow these steps:

1. Login into the **Masking Engine** and select the **Admin** tab.

2. Select the **Edit** icon next to the user you want to edit. The **Edit User** screen will appear with existing user details.

3. Following user information can be modified through the **Edit User** screen:

- First Name

- Last Name

- Email Address

- Password

- Administrator Status

- Welcome Page Status

- Account Status (cannot be changed to *Locked*)

- User Roles (non admin users only)

- User Environments (non admin users only)

4. When you are finished, click **Save**.

✏ **Note**

User's Account Status will be automatically changed to *Locked* on multiple invalid login attempts.

To update a user information using Masking API follow these steps:

1. Access the API client on your Masking Engine, from http://myMaskingEngine.myDomain.com/masking/api-client.

2. Login into the **Masking Engine** and select the **User** endpoint.

3. click **Update user by ID** at the upper right of section and refer to the **Example Value** for parameters required for new users.

4. Enter valid User creation JSON in the **body** section, refer to sample create users JSON.

   Sample User JSON

5. Click on **Execute API Request**.

Deleting A User

To delete a user using the Masking UI follow these steps:

1. Login into the **Masking Engine** and select the **Admin** tab.

2. Select the **Delete** icon next to the user you want to delete.

3. In the confirmation box select **OK**.

To delete a user using Masking API follow these steps:

1. Access the API client on your Masking Engine, from http://myMaskingEngine.myDomain.com/masking/api-client.

2. Login into the **Masking Engine** and select the **User** endpoint.

|

user		Show/Hide	List Operations	Expand Operations
GET	/users	Get all users
POST	/users	Create user
DELETE	/users/{userId}	Delete user by ID
GET	/users/{userId}	Get user by ID
PUT	/users/{userId}	Update user by ID
POST	/users/forgot-password	Send Reset password mail to the user
POST	/users/reset-password	Reset new password for the user

3. Click **Delete user by ID** at the upper right of **/users** section.

|

DELETE	/users/{userId}				Delete user by ID

Parameters

Parameter	Value	Description	Parameter Type	Data Type
userId	(required)	The ID of the user to delete	path	integer

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	Success		
400	Bad request		
401	Unauthorized access		
404	Not found		

Execute API Request

4. Enter the **userID** for the user to be deleted.

5. Click on **Execute API Request**.

Best Practices for Defining Masking Roles

Introduction

The Delphix Masking Engine contains a role definition capability that enables admins to easily create roles for users. This section describes the typical roles and privileges that can be granted to users. It is recommended that the masking administrator implementing these roles consult IT Security and follow existing policies for data access. Roles are added by clicking the appropriate checkboxes within the add role function in the Settings tab. A sample RACI document and examples of roles / privileges are located below.

Roles for operating the Delphix Masking Engine are shared primarily between the masking administration team and the teams that support the applications that will be on-boarded to the Masking Engine. The admin will manage central functions of the engine including definition of custom domains, profiler expressions, algorithms, role and user definitions. The masking Engine is flexible enough to enable application teams with these functions as well, but it is recommended that these shared functions be managed by the admin team. The admin team should have an account registered with Delphix Support and be the main interface for issues and maintenance support from Delphix.

Masking processes can be developed for each application by the central admin team or the individual application teams, often determined by the volume of applications to be on-boarded. The RBAC model employed by Delphix Masking can support different implementation models. Your Delphix support team can assist in constructing roles to meet your needs.

Once roles are defined, they can be assigned to individual user IDs for the environments that those users have responsibility. Administrators will have access to all masking settings and environments by default.

> ✏️ **Note**
>
> 1. Administrator access provides unlimited access to all functions and environments; this role should be granted to the central administration team.
> 2. All privileges is a default role (predefined) which will provide all functions for each environment a user is given access to.
> 3. Connector access should be controlled and administered by personnel responsible for database access.

Sample RACI

Teams: IT Security DM = Data masking admin team Application = App owner/SME DBA = Database admin QA = QA/Test environment owner PM = project management

Role	Description	Accountable	Responsible	Consulted	Informed
Security Policy	Determine data types that are sensitive for the enterprise.	IT Security	IT Security	DM, Application	DBA, QA

Role	Description	Accountable	Responsible	Consulted	Informed
Program Management	Maintain program plan and implementation schedule, tracking and reporting.	PM	DM, Application	QA, IT Security	DBA
Inventory Management	Apply security policy to application schemas/ files.	Application	DM, Application	DBA, QA	IT Security
Data Masking	Build, maintain, schedule masking processes.	Application	DM, DBA	QA	IT Security
Masked Data Validation	Review and approve inventories and masked data.	Application	Application, DBA, QA	DM	IT Security
Masked Data Deployment	Deploy masked data to required environments.	Application	Application, DBA, QA	DM, QA	IT Security
Environment Audit	Assure applications are compliant with masking.	IT Security	IT Security	DM, DBQ, QA	Application
Masking Administration	Manage masking tool central functions, create custom algorithms, domains, profiler expressions, roles, users.	DM	DM	Application, IT Security, DBA	QA

Sample Roles for Masking

Role	Description	*Delphix Masking Functions*
Administrator	Manages masking server updates and upgrades; works with IT Security to update domains, algorithms and profiler expressions / sets.	Unrestricted access to all the engine functions. The Admin role is assigned via the checkbox in the add user page of the UI.

Role	Description	*Delphix Masking Functions*
IT Security Analyst	Determines domains to be masked and high-level method for each domain and communicates them to administrator for inclusion in masking engine, responsible for masking audit functions.	Unrestricted access for all settings functions; access to all application functions except environment and environment create, delete, update.
Application Roles (per environment)		
All Privileges	Super user for an environment.	Unrestricted access for an application environment; central admin or security analyst will determine if this role can modify settings.
DBA	Manages user privileges, database performance and schema definition.	Manage connectors for application database, scripting and scheduling (no settings).
SME / Analyst / Developer	Application subject matter expert, application developer, data analyst, application architecture.	Manage inventories, create, view jobs.
Operations Roles (per environment)		
Operator	Schedule jobs, execute jobs, verify results, run automation scripts.	All job privileges.
Environment Owner	Determine workflow, monitor tool usage for environment.	Approve workflow and inventories, privileges to view for settings and environment.

Audit Logs

Delphix helps you keep a record of user actions taken in the UI or directly through our REST APIs. You can access these audit logs directly from our UI or through our APIs.

Audit Log UI Page

The Audit Log page can be found in the UI under the Audit tab. This page contains information on what action occurred, the user that performed the action, and the time at which the action occurred. It also provides the ability to filter based on:

- user

- time range

- arbitrary search string

- action type or action target, or both (create, connector or create a database connector)

Audit Log APIs

With 5.3.2.0, Delphix introduced an endpoint to get all Audit Logs. This endpoint contains the user name, action type, target, status, start time, and end time. For more information please refer to API documentation.

What Gets Logged?

User actions are categorized into the following:

Cancel	Create	Delete	Edit	Export	Get	Get All
Import	Lock	Login	Logout	Run	Test	Unlock

The objects that user actions target are categorized into the following:

Algorithm	Analytics	Application	Application Log	Async Task	Audit Log	
Column Metadata	Database Connector	Ruleset Connector	Database Ruleset	Domain	Encryption Key	Environment
Execution	File	File	File Field	File Format	File	File Ruleset

		Connector	Download	Metadata		Metadata	
File Upload	LDAP	Mainframe Dataset Connector	Mainframe Dataset Field Metadata	Mainframe Dataset Format	Mainframe Dataset Metadata	Mainframe Dataset Ruleset
Masking Job	Profile Expression	Profile Job	Profile Set	Re Identification Job	Role	SSH Key
SSO	Syncable Object	System Information	Table Metadata	Tokenization	User	

Retention Policy

The default policy stores the last one million Audit Log entries. Any entries older than the most recent million are removed daily. Additionally, there is a fail-safe mechanism that prevents an attacker from forcing an unbounded number of actions to be logged to overload the system's disk space. In the event that such an attack occurs, Delphix also logs it to the application logs.

Recommendation

If a full record of all Audit Log entries is desired, Delphix recommends using the new API to periodically retrieve new entries from the Audit Logs.

Kerberos Configuration

Introduction

As of 5.3.0.0, the Delphix Masking Engine supports Kerberos authentication for Oracle, MS SQL Server, and Sybase connections. Utilizing this service requires the presence of a Kerberos Key Distribution Center (KDC) server as well as additional configuration actions to be done on both the Masking Engine and the database. This document presents configuration instructions for enabling and using Kerberos on the Delphix Masking Engine, as well as reference configurations for enabling Kerberos on the Databases. Although other configurations are possible, the configurations in this document have been validated by Delphix.

Terminology

Throughout this document, the following example values are used. To recreate these reference environments, these values must be replaced with real values appropriate for your network environment: - .bar.com - the DNS domain of the network - BAR.COM - the Kerberos domain - me-host - the hostname of the Masking Engine - foo-kcd - the hostname KDC server - krbuser - the Kerberos principal to be granted access to the database for masking

Configuring Kerberos on the Appliance

This section details the steps required to configure Kerberos on your appliance.

Launch the Delphix Server Setup UI and perform the following steps to enable Kerberos:

a. From the **Network Authorization** widget, click **Modify**.

b. Select the checkbox before **Use Kerberos authentication to communicate with remote hosts** field.

c. Click the plus symbol to add record(s) for your KDCs, and populate other fields appropriately for your network environment. Upon pressing **Save**, your configuration will be tested. If the engine is able to authenticate to the KDC with the supplied configuration, the configuration is applied immediately.

Creating Masking Database Connectors using Kerberos

Once the Delphix Appliance is configured for Kerberos, creating Connectors using Kerberos authentication is simple:

Assuming you are using the same user principal configured in Server Setup, the keytab will be used and it is unnecessary to enter a password in the Connector definition.

For Sybase database Connectors, it is necessary to supply the service principal name as an additional configuration item. For Oracle DB, this value is determined automatically. For MS SQL Server it is determined based on the reverse DNS mapping of the Server Name (refer to the section on MS SQL Server below).

NOTE: If any changes are made to the underlying **krb5.conf** configuration file, these changes will not be recognized by the engine until after the next database connection attempt. Therefore, expect to have to hit "Test Connection" twice after making any changes to the **krb5.conf** file. It does not matter if the first connection attempt succeeds or fails.

Reference Database Configurations

The following are a series of reference Kerberos configuration procedures and troubleshooting notes for the supported databases. These are meant to serve as examples to be further customized according to the user's specific network environment and security needs.

Oracle Database

Overview

This document describes how to set up an Oracle DB instance for kerberized connections. The following steps are described: - Creating a service principal and adding it to the DB system - Configuring the database to use kerberos authentication - Creating DB users identified via kerberos - Troubleshooting tips

Prerequisites

This document assumes you already have a kerberized network environment with an MIT Kerberos KDC. These procedures have been tested successfully with Oracle database versions 11.2.0.2, 11.2.0.4 and 12.2.1. Oracle database version 12.1.0.1 did not work in our testing.

You will need the following from your Kerberos environment: - The krb5.conf file - A user principal and associated password or keytab you'd like to use to log into the database - The ability to create a service principal for the Oracle DB and retrieve the associated keytab

This section of the document uses these example values in addition to those mentioned above: - The oracle database is: ora-db.bar.com. - The oracle service name is: oracle

Creating the Oracle Service Principal

The service principal will be named: /@ Given our default values above, this works out to: oracle/ora-db@bar.com

Notice that the hostname is whatever the database system thinks its hostname is - that is, the output of "uname -n" on the database system, rather than the actual DNS name of the database system. Typically, these values would be the same, but this is not always the case.

On the KDC, run:

```
# kadmin.local
kadmin.local: addprinc -randkey oracle/ora-db@bar.com
kadmin.local: ktadd -norandkey -k /var/tmp/ora-db.keytab oracle/ora-db@bar.com
```

Copy the resulting keytab file (/var/tmp/ora-db.keytab) to the Oracle DB system at this location: /etc/v5srvtab

As root on the Oracle DB system, ensure that the keytab has the correct permissions:

```
# chown root:oinstall /etc/v5srvtab
```

```
# chmod 440 /etc/v5srvtab
```

Finally, this is a good opportunity to copy /etc/krb5.conf from the KDC to /etc/krb5.conf on the Oracle DB system. This file should be readable by all users.

Configuring the Oracle Database for Kerberos

Log into the Oracle DB system as the appropriate use for the database in question.

```
$ cd $ORACLE_HOME
$ vi network/admin/sqlnet.ora
```

Add the following for Oracle 11:

```
SQLNET.KERBEROS5_CONF=/etc/krb5.conf
SQLNET.AUTHENTICATION_SERVICES=(BEQ,KERBEROS5)
SQLNET.KERBEROS5_CONF_MIT=true
SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=oracle
```

Or the following for Oracle 12:

```
NAMES.DIRECTORY_PATH=(TNSNAMES, EZCONNECT, HOSTNAME)
SQLNET.KERBEROS5_CONF=/etc/krb5.conf
SQLNET.AUTHENTICATION_SERVICES=(BEQ,KERBEROS5PRE,KERBEROS5)
SQLNET.KERBEROS5_CONF_MIT=true
SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=oracle
```

If the database is Oracle 11 (not necessary on Oracle 12): `$ vi dbs/init.ora` Add this line at the end:
`OS_AUTHENT_PREFIX=""`

Creating a DB User Identified via Kerberos

Log into the Oracle DB system as the appropriate database user and open a database session as the DBA:

```
$ sqlplus / as sysdba
```

On Oracle 12, you may wish to alter your session to create the user in one of the PDBs: `SQL> alter session set`
`container=MYPDB;`

Create the user that will connect to the DB using kerberos:

```
SQL> create user krbdbuser identified externally as 'krbuser@BAR.COM';
```

Grant the user privileges necessary for masking.

This example grants all privileges for the sake of simplicity:

Oracle 11:

```
SQL> grant all privilege to krbdbuser;
```

Oracle 12: (Customize permissions as necessary for your environment).

`SQL> grant connect,resource to krbdbuser;

SQL> grant create tablespace, drop tablespace to krbdbuser;

SQL> grant create table to krbdbuser;

SQL> grant create sequence to krbdbuser;

SQL> grant select_catalog_role to krbdbuser;

SQL> grant unlimited tablespace to krbdbuser;

SQL> grant select_catalog_role to krbdbuser;

SQL> grant alter system to krbdbuser;

SQL> grant sysoper to krbdbuser;

SQL> grant dba to krbdbuser;`

Troubleshooting Tips

- Connecting via JDBC with Kerberos authentication from Delphix Masking involves two steps - a Kerberos login, followed by JDBC connect. A failure stack with an error in the login function indicates a misconfiguration on either the engine or KDC - the engine hasn't even attempted to communicate with the database at that point. Failure stacks are saved in the debugging log for masking.

- Login exceptions that mention a checksum error mean either the password or keytab supplied doesn't match the expected password/key on the KDC for the principal you're trying to use. Server Setup verifies that your keytab works at configuration time, but it could stop working if the key for your principal is updated on the KDC.

- Prior to version 12, Oracle databases instances assume they can create/write a particular temporary file to store Kerberos credentials for the DB. This means if you attempt to run multiple kerberized instances of Oracle 11 on the same system or VM, and the databases run as different system users, the first Oracle instance that performs Kerberos auth will create and own this file. Kerberos authentication will fail to function on all other instances.

MS SQL Server

Overview

This is an overview of the step necessary to get your Masking Engine talking to a MS SQL Server database using kerberos authentication. Since Active Directory already uses Kerberos for authentication, little or no additional configuration is need on the MS SQL Database server.

The following steps are described in this section: - Create the necessary SPNs (Service Principal Names) for your MSSQL Database service in AD - Create the DB Connector on the masking engine - Creating a keytab for an AD User - Troubleshooting tips

Prerequisites

Configuring cross-realm trust between Active Directory and an MIT KDC Server is a complex topic, and will not be described here. In the absence of such a setup, it is possible to make the Delphix Appliance a Kerberos client of the Active Directory (AD) Server. In this configuration, no additional KDC in necessary. The example below assumes this kind of configuration.

This section of the document uses these example values in addition to or instead of those mentioned above: - The MSSQL server database is named mssql-db.bar.com. - The AD user configured for masking access to the MSSQL database is aduser (rather than krbuser in other examples elsewhere in this document). - The AD user that start the MS SQL Server service on the DB Server is dbuser.

Creating SPNs for the Database Service

MS SQL Server service will typically register several SPNs with AD upon startup. However, there are several conditions which can cause these SPNs to not be registered successfully, or to be registered with service names other than those that are expected by the Microsoft JDBC Driver for SQL Server employed by Delphix Masking.

The service principal name for an MS SQL Server expected by Delphix Masking is: MSSQLSvc/: For example, the SPN for our example MS SQL Server would be: `MSSQLSvc/mssql-db.bar.com:1433`

In addition, it is **required** that a reverse mapping exist in DNS from the IP address of the MS SQL Server system to the FQDN registered.

The following commands may be run in PowerShell on the MS SQL Server to assist in debugging SPN related issues:

List all SPNs for dbuser:

```
setspn -L -U dbuser
```

Deleting an old SPN associated with dbuser:

```
setspn -U -D MSSQLSvc/other-server.ad.bar.com:SQL2008R2 dbuser
```

Here's how to create the SPN describe above:

```
setspn -U -S MSSQLSvc/mssql-db.bar.com:1433 dbuser
```

Creating the Database Connector on the Masking Engine

Once the above steps are complete, creating the database connector can be performed using the procedure above. Enter the username and optionally, password of the AD user in the Connector definition. Be sure that the AD user has sufficient access to the MS SQL Database for masking.

The password field can be left blank when creating the connector if the user is the same user configured in Server Setup for the appliance. Since keytabs are not typically used in an AD environment, it may be useful to create one manually, to avoid having a password in the DB Connector.

Creating a keytab file for an AD user

On a unix or MAC system with MIT Kerberos CLI utilities installed:

```
# ktutil

ktutil: addent -password -p krbuser -k 1 -e arcfour-hmac

<type password for krbuser>

ktutil: addent -password -p krbuser -k 1 -e aes128-cts-hmac-sha1-96

<type password for krbuser>

ktutil: addent -password -p krbuser -k 1 -e aes256-cts-hmac-sha1-96

<type password for krbuser>

ktutil: write_kt /var/tmp/krbuser.keytab

ktutil: exit

# base64 /var/tmp/krbuser.keytab ;# This is string to user for keytab in Server Setup kerberos configuration
```

Note kvno doesn't matter when using Kerberos keytabs with AD. The password must match the active password for the AD user in question.

Troubleshooting Tips

The client uses the incorrect service name. This will typically manifest an exception mentioning cred, like:

```
Caused by: org.ietf.jgss.GSSException: No valid credentials provided (Mechanism level: Fail to create
credential. (63) - No service creds)

at sun.security.jgss.krb5.Krb5Context.initSecContext(Krb5Context.java:770)

at sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:248)

at sun.security.jgss.GSSContextImpl.initSecContext(GSSContextImpl.java:179)

at com.microsoft.sqlserver.jdbc.KerbAuthentication.intAuthHandShake(KerbAuthentication.java:163)
        ... 101 common frames omitted

Caused by: sun.security.krb5.internal.KrbApErrException: Fail to create credential. (63) - No service creds  at
sun.security.krb5.internal.CredentialsUtil.acquireServiceCreds(CredentialsUtil.java:162)

at sun.security.krb5.Credentials.acquireServiceCreds(Credentials.java:458)

at sun.security.jgss.krb5.Krb5Context.initSecContext(Krb5Context.java:693)
        ... 104 common frames omitted
```

Why might this happen: - You're using the JTDS JDBC driver, and your MSSQL Server's IP address doesn't have a reverse mapping in DNS. In this case, the driver may construct a service name like: MSSQLSvc/: and try to use that. Either correct DNS to have a valid reverse mapping for the IP of your SQL server, or manually add an SPN to the active directory for the name the JDBC client is trying to use: - Determine the user that starts MSSQL Server on your DB machine. - From PowerShell, do: setspn -AU MSSQLSvc/:1433 Example: setspn -AU MSSQLSvc/10.43.100.101:1433 AD\dbuser - The database server has multiple DNS names (FQDNs). In this case, SPNs may be registered only for some of them. It may be necessary to add SPNs for the other FQDNs as above. - The MS SQL Server didn't automatically register an SPN. There is a limit (in the thousands) to the number of SPNs that may be registered for a given AD user. It is quite possible to hit this limit in an environment where many MS SQL Server VMs are actively created and destroyed with the same configuration.

Note In Active Directory, setspn isn't creating a service principal with distinct key as is typical for services on MIT KDCs - rather it's mapping the service principal to the key for the AD user in question.

The SPN for the SQL Server is registered to the incorrect AD account

Manifests as an exception with this text: GSS failure: Defective token detected (Mechanism level: AP_REP token id does not match!)

Resolution: From PowerShell on the MS SQL Server:

```
PS> setspn -Q <SPN>
```

This will show what the user has the SPN registered.

```
PS> setspn -U -D <SPN> <WRONG_ACCT>
```

This will unregister the SPN from that user

```
PS> setspn -AU <SPN> <CORRECT_ACCT>
```

Sybase

Creating a principal and corresponding keytab on the KDC

1. SSH into the KDC as the user with sufficient privileges to run kadmin.local

2. Run the Kerberos configuration CLI with kadmin.local

3. Add a new principal you want to authenticate as later with: `add_principal <principalName>`

   We're going to continue to use **krbuser** as our example Kerberos principal.

4. Once you've created the principal and provided it a password, we need to generate a keytab for it. Do so via the following command:

```
ktadd -norandkey -k v5srvtab krbuser
```

In this case, v5srvtab is the keytab filename, and it will be placed into whatever directory you've invoked kadmin.local from. Presumably, this will be the home directory of the machine.

1. You now have everything you need done on the KDC, but you will need your keytab file later as well as the **krb5.conf** file that is located in the home directory of the KDC, so consider moving them somewhere (probably your local machine) that will be convenient for you to access later.

Configuring the Sybase image for Kerberos

1. Startup a Sybase database.

2. **Note**: Each Sybase database machine may have multiple Sybase instances running on it at a given point in time. In this case, I am configuring the ASE_1550_S5 instance, but these steps can be done on any instance so long as you change the $SYBASE_HOME directories accordingly.

3. Connect to the particular Sybase instance you are working on and invoke the following sql statement:

```
sp_configure 'use security services', 1
```

1. Continue to create a user with the same name as the principal name you created previously on the KDC, in this case **krbuser**:

   ```
   sp_addlogin krbuser, <password>
   ```

2. Change your **$SYBASE** environment variable to point to the Sybase directory for whichever instance you are configuring. In this case, we want to do:

```
export SYBASE=/opt/sybase/15-5
```

1. Open the **$SYBASE/interfaces file**, and find the header for whichever Sybase instance you are configuring. In our case, it is **ASE_1550_S5**. You should see something that looks like this:

```
ASE1550_S5
```

```
 `master tcp ether 10.43.89.241 5500`

 `master tcp ether localhost 5500`

 `query tcp ether 10.43.89.241 5500`

 `query tcp ether localhost 5500`
```

You want to add the following line to this:

```
secmech 1.3.6.1.4.1.897.4.6.6
```

```
 This line is static, while the other lines in this section are dynamically generated for your instance. So,
 your final result should look something like this:
```

```
ASE1550_S5
```

`master tcp ether 10.43.89.241 5500` **< your numbers will vary**

`master tcp ether localhost 5500` **< your numbers will vary**

`query tcp ether 10.43.89.241 5500` **< your numbers will vary**

`query tcp ether localhost 5500` **< your numbers will vary**

1. Navigate to **$SYBASE/OCS-15_0/config**. You should see **libtcl64.cfg** and **libtcl.cfg**

a. Change the contents of **libtcl64.cfg** to be this:

```
 `[DIRECTORY]`

 `;ldap=libsybdldap.so ldap://ldaphost/dc=sybase,dc=com`

 `[SECURITY]`
 `csfkrb5=libsybskrb64.so secbase=@bar.com libgss=/lib64/libgssapi_krb5.so.2.2
```

 [FILTERS] ;ssl=libsybfssl.so`

b. Change the contents of **libtcl.cfg** to be this:

```
   `[DIRECTORY]`

   `;ldap=libsybdldap.so ldap://ldaphost/dc=sybase,dc=com`

   `[SECURITY]`

   `csfkrb5=libsybskrb.so secbase=@bar.com libgss=/lib64/libgssapi_krb5.so.2.2`

   `[FILTERS]`

   `;ssl=libsybfssl.so`
```

c. **Note** that the @bar.com value is our realm name that is determined by the KDC. Realistically, you should never have to deal with this, and it should never change, but if for some reason it does, that value needs to be updated.

1. Create a directory for those Kerberos config files you created on the KDC in the previous set of steps:

```
sudo mkdir /krb
```

Copy into /krb your keytab file **v5srvtab** and config file **krb5.conf** that you took off of the KDC earlier.

1. Head to **$SYBASE/ASE-15_0/install** and open the **RUN_ASE1550_S5** file. We're going to add information so that Sybase knows where to find our keytab and our krb5.conf file, so change the content to look like this:

```
#!/bin/sh

#

# ASE page size (KB) :     4096

# Master device path:    /opt/sybase/devices/data5/S5_master.dat

# Error log path:         /opt/sybase/errorlogs/ASE1550_S5.log

# Configuration file path:      /opt/sybase/15-5/ASE-15_0/ASE1550_S5.cfg

# Directory for shared memory files:     /opt/sybase/15-5/ASE-15_0

# Adaptive Server name: ASE1550_S5

#

export **KRB5_KTNAME**=/krb/v5srvtab

export **KRB5_CONFIG**=/krb/krb5.conf

/opt/sybase/15-5/ASE-15_0/bin/dataserver \

-kASE1550_S5@bar.com \

-d/opt/sybase/devices/data5/S5_master.dat \

-e/opt/sybase/errorlogs/ASE1550_S5.log \

-c/opt/sybase/15-5/ASE-15_0/ASE1550_S5.cfg \

-M/opt/sybase/15-5/ASE-15_0 \

-sASE1550_S5 \
```

1. Reboot the Sybase instance you're working so that it reads in all of these configuration changes.

2. Connect to the Sybase instance as the **dbo** user so that you may give dbo privileges to your Kerberos authentication login on a particular database within the instance. Below is an example of doing so with the database **potatoes**:

```
>> sql5
```

```
1> use potatoes
```

```
2> go
```

```
1> sp_addalias instructions, dbo
```

```
2> go
```

```
Alias user added.
```

```
(return status = 0)
```

1. Now, to access the Sybase instance via Kerberos and confirm success, you can do the following set of commands (I put these three lines into a script called **connect.sh** for future convenience):

```
#!/bin/sh
```

```
kinit -k -t /krb/v5srvtab <yourPrincipalName>
```

```
export SYBASE='/opt/sybase/15-5'
```

```
/opt/sybase/15-5/OCS-15_0/bin/isql64 -V -SASE1550_S5
```

Testing by creating a Kerberos Connector on the Delphix Engine

1. Start by configuring your engine for Kerberos. SSH into the engine as the Delphix user and run the following command:

   ```
   /opt/delphix/server/bin/jmxtool tunable set enabled_features KERBEROS true
   ```

2. Log into the Delphix Engine and proceed through the first-time setup.

3. Once the first-time setup is complete, log into the Delphix Setup page, proceed to Preferences > Kerberos Configuration. Add the information for your KDC to configure it with the principal name you created earlier, **krbuser**. You can get the keytab by running the following command on your keytab file:

base64 v5srvtab

Copy the output as plaintext into the keytab field of the Kerberos configuration box.

Finally, create a Sybase connector with parameters that look like this, and if your "test connection" attempt succeeds you're all set!

Create Connection

Type

Database - Sybase ▾

◉ Basic   ○ Advanced

Connection Name

Sybase kerberos

Schema Name

dbo

Database Name

potatoes

Host Name/ IP

sybaseHostName.bar.com

Port

4000

☑ Use Kerberos Authentication

Principal Name

krbuser

Service Principal

ASE1550_S5

Password

LEAVE BLANK TO USE KEYTAB

[Test Connection]   [Cancel] [Save]

DB2 Connector License Installation

If you have been licensed to use the Delphix Masking DB2 Connector for Mainframe or DB2 Connector for iSeries, you will need to obtain the respective DB2 Connector package (tar file) and apply it to your Masking Engine(s). Each package is intended to be installed and run from a workstation or laptop, not from the Delphix Appliance. These packages contain a script that must be used in a bash shell and depends on the availability of the **curl** and **ssh** commands to install the respective license on your remote Delphix Appliance.

Applying DB2 Connector for Mainframe

1. Go to
   https://download.delphix.com/folder/580/Delphix%20Product%20Releases/DB2%20Masking%20Mainframe
   and download DB2MaskingMainframe.tar

2. Extract its contents using tar -xvf DB2MaskingMainframe.tar

3. cd db2-license

4. ./installdb2license.sh -h MASKING_ENGINE_HOST -P MASKING_ENGINE_PORT -u
   MASKING_ENGINE_ADMIN_USERNAME -p MASKING_ENGINE_ADMIN_PASSWORD [-C
   MASKING_ENGINE_PUBLIC_KEY_FILE]

Where:

MASKING_ENGINE_HOST is the hostname for where the masking engine is running.

MASKING_ENGINE_PORT is the port for where the masking engine is listening on the MASKING_ENGINE_HOST (default is port 80).

MASKING_ENGINE_ADMIN_USERNAME is the username for connecting to the masking engine (e.g., delphix_admin).

MASKING_ENGINE_ADMIN_PASSWORD is the masking engine password for .

MASKING_ENGINE_PUBLIC_KEY_FILE is the optional trusted server certificate (server public key) obtained from the masking engine.

> ✏️ **Note**
>
> To run the enablement script securely, run installdb2license.sh specifying your secure port (e.g., 8443) and trusted server certificate (server public key) using the -C option.

The script will enable the DB2 Mainframe connector and then recycle the Masking Engine, prompting the user for the Delphix sysadmin password for to first stop the Masking Engine and then to start it. After the DB2MaskingMainframe.tar package has been applied to your Masking Engine(s), "Database - MAINFRAME DB2" will appear in the Connector drop-down of the Masking Engine UI and can be used in the same way as other Database Connectors to create, profile, mask, certify, and provision rulesets.

Applying DB2 Connector for iSeries

1. Go to https://download.delphix.com/folder/585/Delphix%20Product%20Releases/DB2%20Masking%20i-Series and download DB2MaskingISeries.tar

2. Extract its contents using tar -xvf DB2MaskingISeries.tar

3. cd db2-license

4. ./installdb2license.sh -h MASKING_ENGINE_HOST -P MASKING_ENGINE_PORT -u MASKING_ENGINE_ADMIN_USERNAME -p MASKING_ENGINE_ADMIN_PASSWORD [-C MASKING_ENGINE_PUBLIC_KEY_FILE]

Where:

MASKING_ENGINE_HOST is the hostname for where the masking engine is running.

MASKING_ENGINE_PORT is the port for where the masking engine is listening on the MASKING_ENGINE_HOST (default is port 80).

MASKING_ENGINE_ADMIN_USERNAME is the username for connecting to the masking engine (default is delphix_admin).

MASKING_ENGINE_ADMIN_PASSWORD is the masking engine password for MASKING_ENGINE_ADMIN_USERNAME.

MASKING_ENGINE_PUBLIC_KEY_FILE is the optional trusted server certificate (server public key) obtained from the masking engine.

> ✏️ **Note**
>
> To run the enablement script securely, run installdb2license.sh specifying your secure port (e.g., 8443) and trusted server certificate (server public key) using the -C option.

The script will enable the DB2 iSeries connector and then recycle the Masking Engine, prompting you for the Delphix sysadmin password for to first stop the Masking Engine and then to start it. After the DB2MaskingISeries.tar package has been applied to your Masking Engine(s), "Database - ISeries DB2" will appear in the Connector drop-down of the Masking Engine UI and can be used in the same way as other Database Connectors to create, profile, mask, certify, and provision rulesets.

Masking Engine Icon Reference

This topic illustrates the icons that appear on the Delphix Masking Engine graphic user interface and describes the meaning of each.

Icon	Description
✏️	Edit
📤	Export
📋	Copy
❌	Delete
⭐	Job Success
📄	Job Created
🔵	Mask
▶️	Run Job
🔄	Ruleset Refresh
N/A	Ruleset refresh not applicable for file rulesets
🕐	Job Running
⬟	Cancel Job
⏳	Ruleset Refresh in Progress

Delphix Masking Terminology

Before getting started with the Delphix Masking Engine, an overview of universal terms and concepts will build and unify how different masking components come together. The following provides a brief overview of the key concepts within the masking service.

High Level Concepts

These concepts are the high level concepts users run into.

Term	Definition
Application	An Application is a tag that is assigned to one or more environments. We recommend using an application name that is the same as the application associated with the environments.
Connector	Connectors are any set of data (database, file, etc) that have been connected to the Delphix Data Platform. These data sources can be physical or virtualized data sources.
Domain	A domain represents a correlation between various sensitive data categories (social security numbers) and the way it should be secured.
Environment	An environment is a construct that can be used to describe a collection of masking jobs associated with a group of data sources.
In-place	In-place masking is 1 of 2 procedures that can be used to apply masking algorithms to a data source. By choosing the In-place option, Delphix will read data from the data source, secure the data in the Engine and then update the data source with the secure data.
On-the-fly	On-the-fly masking is the second procedure that can be used to apply masking algorithms to a data source. By choosing the On-the-fly option, Delphix will read data from the data source, secure the data in the Engine and then place the secure data in a target source (different from the location of the original data source).
Inventory	An inventory describes all of the data present in a particular data source and defines the methods which will be used to secure it. Inventories typically include the table name, column name, the data classification, and the chosen algorithm.
Profile	Profiling uses a variety of different methods to classify data in a data source into different categories. These categories are known as domains. The profile process also assigns recommended algorithms for securing the data based on the the domain.

Term	Definition
Ruleset	A rule set is group of tables or flat files within a particular data source that a user may choose to run profile, masking, or tokenization jobs on.

Masking Algorithms

The following terminology is around the different Algorithms that users may use to secure their data.

Term	Definition
Algorithm Framework	A type of masking algorithm. One or more usable instances of an *algorithm framework* may be created. For example, "FIRST NAME SL" is an instance of the Secure Lookup *algorithm framework*.
Algorithm Instance	A named combination of algorithm framework and configuration values. *Algorithm instances* are applied to data fields and columns in the inventory in order to mask data.
Built-in Algorithm	An algorithm instance or framework included with the Masking Engine software. This includes several built-in algorithm instances that provide masking behavior that doesn't correspond to any built-in algorithm framework.
Custom Algorithm	An algorithm instance or framework not included with the Masking Engine software. *Custom algorithms* may be added to the Masking Engine by an administrator.
Non-conformant Data	Some masking algorithms require data to be in a particular format. The required format may vary by the configuration of the algorithm instance. For example, a particular Segment Mapping algorithm might be configured to expect a 10 digit number. Data which doesn't fit the pattern expected by an algorithm is called *nonconforming data* or *non-conformant data*. By default, *non-conformant data* is not masked, and warnings are recorded for the masking job. Warnings are indicated by a yellow triangle warning marker next to the job execution in Environment and Job Monitor pages. Whether *non-conformant data* results in a warning or failure is configurable for each algorithm instance.
Collision	The term *collision* describes the case where a masking algorithm masks two or more unique input values to the same output value. For example, a first name Secure Lookup algorithm might mask both "Amy" and "Jane" to the same masked value "Beth". This may be desirable, in the sense that it further obfuscates the original data, however *collisions* are problematic for data columns with uniqueness constraints.
Secure Lookup	The most commonly used algorithm framework. Secure lookup works by replacing each data value with a new value chosen from an input file. Replacement values are chosen based on a cryptographic hash of the original value, so masking output is consistent for each input. Secure lookup algorithms are easy to configure and work with different languages.

Term	Definition
	When this algorithm replaces real data with fictional data, *collisions*, described above, are possible. Because many types of data, such as first or last name, address, etc, are not unique in real data, this is often acceptable. However, if unique masking output for each unique input is required, consider using a mapping or segment mapping algorithm, described below.
Segment Mapping	This algorithm permutes short numeric or alpha-numeric values to other values of the same format. This algorithm is guaranteed to not produce collisions, so long as the set of permissible mask values is at least as large as input or "real" set. The maximum number of digits or characters in the masked value is 36. You might use this method if you need columns with unique values, such as Social Security Numbers, primary key columns, or foreign key columns.
Mapping	Similar to secure lookup, a mapping algorithm allows you to provide a set of values that will replace the original data. There will be no collisions in the masked data, because each input is always matched to the same output, and each output value is only assigned to one input value. In order to accomplish this, the algorithm records, in an encrypted format, all known input to output mappings. You can use a mapping algorithm on any set of values, of any length, but you must know how many values you plan to mask, and provide a set of unique replacement values sufficient to replace each unique input value. NOTE: When you use a mapping algorithm, you cannot mask more than one table at a time. You must mask tables serially.
Binary Lookup	Replaces objects that appear in object columns. For example, if a bank has an object column that stores images of checks, you can use binary lookup algorithm to mask those images. The Delphix Engine cannot change data within images themselves, such as the name on X-rays or driver's licenses. However, you can replace all such images with a new, fictional image. This fictional image is provided by the owner of the original data.
Tokenization	The only type of algorithm that allows you to reverse its masking. For example, you can use a tokenization algorithm to mask data before you send it to an external vendor for analysis. The vendor can then identify accounts that need attention without having any access to the original, sensitive data. Once you have the vendor's feedback, you can reverse the masking and take action on the appropriate accounts. Like mapping, a tokenization algorithm creates a unique token for each input such as "David" or "Melissa." The Delphix Engine stores both the token and original so that you can reverse masking later.
Min Max	Values that are extremely high or low in certain categories allow viewers to infer someone's identity, even if their name has been masked. For example, a salary of $1 suggests a company's CEO, and some age ranges suggest higher insurance risk. You can use a min max algorithm to move all values of this kind into the midrange.
Data Cleaning	Does not perform any masking. Instead, it standardizes varied spellings, misspellings, and abbreviation for the same name. For example, "Ariz," "Az," and "Arizona" can all be cleaned to "AZ."

Term	Definition
Free Text Redaction	Helps you remove sensitive data that appears in free-text columns such as "Notes." This type of algorithm requires some expertise to use, because you must set it to recognize sensitive data within a block of text. One challenge is that individual words might not be sensitive on their own, but together they may be. This algorithm uses profiler sets to determine which information it needs to mask. You can decide which expressions the algorithm uses to search for material such as addresses. For example, you can set the algorithm to look for "St," "Cir," "Blvd," and other words that suggest an address. You can also use pattern matching to identify potential sensitive information. For example, a number that takes the form 123-45-6789 is likely to be a Social Security Number. You can use free text redaction algorithm to show or hide information by displaying either a "deny list" or an "allow list."

Profile Job Concepts

The following set of concepts are options available to the user for configuring a profiling job.

Term	Definition
Job Name	A free-form name for the job you are creating. Must be unique.
Multi-Tenant	Check the box if the job is for a multi-tenant database. This option allows existing rulesets to be reused to mask identical schemas via different connectors. The connector can be selected at job execution time.
Rule Set	Select a ruleset that this job will execute against.
No. of Streams	The number of parallel streams to use when running the jobs. For example, you can select two streams to run two tables in the ruleset concurrently in the job instead of one table at a time.
Min Memory (MB) *optional*	Minimum amount of memory to allocate for the job, in megabytes.
Max Memory (MB) *optional*	Maximum amount of memory to allocate for the job, in megabytes.
Feedback Size *optional*	The number of rows to process before writing a message to the log. Set this parameter to the appropriate level of detail required for monitoring your job. For example, if you set this number significantly higher than the actual number of rows in a job, the progress for that job will only show 0 or 100%

Term	Definition
Profile Sets *optional*	The name of a profile set, which is a subset of expressions (for example, a subset of financial expressions).
Comments *optional*	Add comments related to this job.
Email *optional*	Add email address(es) to which to send status messages. Separate addresses with a comma (,).

Masking Job Concepts

These concepts are options available to the user for configuring a masking job.

Term	Definition
Job Name	A free-form name for the job you are creating. Must be unique across the entire application.
Masking Method	Select either In-Place or On-The-Fly.
Multi-Tenant	Check the box if the job is for a multi-tenant database. This option allows existing rulesets to be reused to mask identical schemas via different connectors. The connector can be selected at job execution time.
Rule Set	Select a ruleset for this job to execute against.
Masking Method	Select either In-place or On-the-fly.
Min Memory (MB) optional	Minimum amount of memory to allocate for the job, in megabytes.
Max Memory (MB) optional	Maximum amount of memory to allocate for the job, in megabytes.
Update Threads	The number of update threads to run in parallel to update the target database. For database using T-SQL, multiple update/insert threads can cause deadlock. If you see this type of error, reduce the number of threads that you specify in this box.
Commit Size	The number of rows to process before issuing a commit to the database.

Term	Definition
Feedback Size	The number of rows to process before writing a message to the logs. Set this parameter to the appropriate level of detail required for monitoring your job. For example, if you set this number significantly higher than the actual number of rows in a job, the progress that job will show 0% or 100%.
Disable Trigger *optional*	Whether to automatically disable database triggers. The default is for this check box to be clear and therefore not perform automatic disabling of triggers.
Drop Index *optional*	Whether to automatically drop indexes on columns which are being masked and automatically re-create the index when the masking job is completed. The default is for this check box to be clear and therefore not perform automatic dropping of indexes.
Prescript *optional*	Specify the full pathname of a file that contains SQL statements to run before the job starts, or click Browse to specify a file. If you are editing the job and a pre script file is already specified, you can click the Delete button to remove the file. (The Delete button only appears if a prescript file was already specified.)
Postscript *optional*	Specify the full pathname of a file that contains SQL statements to be run after the job finishes, or click Browse to specify a file. If you are editing the job and a postscript file is already specified, you can click the Delete button to remove the file. (The Delete button only appears if a postscript file was already specified.)
Comments *optional*	Add comments related to this masking job.
Email *optional*	Add email address(es) to which to send status messages.

Preparing Data

Database User Permissions for executing Masking and Profiling Jobs

Introduction

This section provides the recommended list of permissions required for executing Masking and Profiling jobs on the Delphix Masking Engine. This page provides general permission recommendations. The subsequent pages in this section provide detailed recommendations for specific databases.

Recommend creating a separate Database user (i.e. named *Masking User*) to be created across all the databases with the appropriate permissions on the schemas to be masked. If needed create multiple users. The appropriate permissions for the database *Masking User* are listed below.

The benefits of having a separate DB *Masking User*:

- Replicating the new user (and privileges) are easier

- Access Audits are much easier

- Can be created as a central AD user and used at many places simultaneously

List of Database Entitlements Required to Run Masking Jobs

- Read data from Tables

- Write data to Tables

- Update data in tables

- Create indexes

- Drop indexes

- Create triggers

- Drop triggers

- Disable triggers

- Enable triggers

- Alter tables add column

- Alter table delete column

- Create constraints

- Delete constraints

- Disable constraints

- Enable constraints

List of Database Entitlements Required to Run Profiling Jobs

- View Definition (Schema)

- Read Data from Tables

Preparing Oracle Database for Profiling/Masking

Before masking your data, it is important to prepare your database. This section explains the required changes, reasons for the changes, and instructions on how to make the changes.

Archive Logging

What is Archive Logging?

Oracle Database lets you save filled groups of redo log files to one or more offline destinations, known collectively as the archived redo log, or more simply the archive log. The process of turning redo log files into archived redo log files is called archiving. This process is only possible if the database is running in ARCHIVELOG mode. You can choose automatic or manual archiving.

Why is it important to make this change?

Archive logging will slow down masking processes and absorb CPU resources that could be used by the masking process. Furthermore, since masking will change every row in every table being masked logs are only needed for short term recovery and transaction backout.

The choice of whether to enable the archiving of filled groups of redo log files depends on the availability and reliability requirements of the application running on the database. If you cannot afford to lose any data in your database in the event of a disk failure, use ARCHIVELOG mode. The archiving of filled redo log files can require you to perform extra administrative operations.

How exactly do I make this change? (exact commands, etc).

```
ALTER DATABASE NOARCHIVELOG;
```

DB/VDB Memory Allocation

What is SGA? A system global area (SGA) is a group of shared memory structures that contain data and control information for one Oracle database instance. If multiple users are concurrently connected to the same instance, then the data in the instance's SGA is shared among the users. Consequently, the SGA is sometimes called the shared global area.

An SGA and Oracle processes constitute an Oracle instance. Oracle automatically allocates memory for an SGA when you start an instance, and the operating system reclaims the memory when you shut down the instance. Each instance has its own SGA.

The SGA is read/write. All users connected to a multiple-process database instance can read the information contained within the instance's SGA, and several processes write to the SGA during the execution of Oracle. When automatic SGA memory management is enabled, the sizes of the different SGA components are flexible and can adapt to the needs of a workload without requiring any additional configuration. The database automatically distributes the available memory among the various components as required, allowing the system to maximize the use of all available SGA memory. Make sure the DB/VDB memory allocation is sufficient for the workload. Delphix's best practices for sizing a VDB will handle most masking requirements. If you plan to run many concurrent masking jobs a small memory allocation will negatively impact the performance of the masking jobs.

Why is it important to make this change?

To assure that masking jobs will perform at an optimum level.

How exactly do I make this change? (exact commands, etc). Set automatic SGA memory management to enabled. If not allowed set the SGA based on the diagnosis from the AWR report generated during a masking job. The DBA is best suited to make the appropriate tuning changes to the SGA parameters for the version of Oracle being masked.

Undo Tablespace Size And Undo Retention Time:

What is tablespace? Every Oracle Database must have a method of maintaining information that is used to roll back or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. These records are collectively referred to as undo.

Undo records are used to: - Roll back transactions when a ROLLBACK statement is issued - Recover the database - Provide read consistency - Analyze data as of an earlier point in time by using Oracle Flashback Query - Recover from logical corruptions using Oracle Flashback features

When a ROLLBACK statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During database recovery, undo records are used to undo any uncommitted changes applied from the redo log to the datafiles. Undo records provide read consistency by maintaining the before image of the data for users who are accessing the data at the same time that another user is changing it.

Why is it important to make this change?

The masking Engine updates or inserts masked data in batches. In the case of an insert, it only requires the current transaction size for the commit of each table being masked. The default per table stream is 10k rows. However, with an update, the transaction is not complete until the entire table is masked. So, the more tables and more rows and the wider (size) each row is in each table, the more undo space is needed to complete the transaction. Large tables, such as DW tables or history and Audit tables, most often need an increase to the Undo space and undo Retention time for updates. If space or time is exceeded then the masking job may fail with an ORA-01555, Snapshot too old error.

How exactly do I make this change? (exact commands, etc).

It is highly recommended to increase the Undo space and undo Retention time when running in-place jobs on large tables. A general rule of thumb is 2 or 3 times the size of the larges table(s), or if there are multiple tables running at the same time, then all tables combined. A DBA is best suited to make the necessary UDNO Space and UNDO Retention changes.

Redo Logs Are Optimally Sized

What is Redo Logs?

The most crucial structure for recovery operations is the redo log, which consists of two or more preallocated files that store all changes made to the database as they occur. Every instance of an Oracle Database has an associated redo log to protect the database in case of an instance failure.

Why is it important to make this change?

The most important reason to make this change is to keep performance optimal. If redo logs are too small, then the log switching will occur too often, using up valuable Oracle resources.

How exactly do I make this change? (exact commands, etc).

A DBA is best suited to make these changes appropriately.

Change PCTFREE to 40-50:

What is PCTFREE?

PCTFREE and PCTUSED are used together, but PCTFREE is critical for updates. The larger the PCTFREE value the more updates can be done.

Why is it important to make this change?

PCTFREE aids in performance increases for updating Oracle during masking. The Masking Engine does many updates at the same time in batch mode. The more that can be done without DB overhead the faster the masking jobs run.

How exactly do I make this change? (exact commands, etc).

A DBA is best suited to make these changes.

Change Primary Key To ROWID:

What is ROWID?

For each row in the database, the ROWID pseudocolumn returns the address of the row. Oracle Database rowid values contain information necessary to locate a row.

Why is it important to make this change?

This is especially important in masking for performance. IF ROWID is used then Oracle will manage the updates for the rows it tracks using ROWID. This makes updates much faster. On occasion, there may be a key (PK/FK/UK) or ID column with an index that is faster, but generally, ROWID is the fastest.

How exactly do I make this change? (exact commands, etc).

Add ROWID as the logical key on each table in the ruleset using the Masking Engine GUI. Also, in a script you should drop foreign keys, and if possible indices and disable triggers and recreate them after the masking job has been run for any of these types of columns being masked.

Preparing SQL Server Database for Profiling and Masking

Before masking your data, it is important to prepare your database. This section explains the required changes, reasons for the change, and the instructions to make the change.

Logging

What is Simple Recovery Model?

SQL Database Simple Recovery model - Automatically reclaims log space to keep space requirements small, essentially eliminating the need to manage the transaction log space. Operations that require transaction log backups are not supported by the simple recovery model.

Why is it important to make this change?

Reducing the overhead of the transaction logging and the size of the files before checkpoints increases the masking speed significantly.

How exactly do I make this change?

Either (a) use SQL Server Management Studio to open the DB properties dialog box and select the "simple recovery model" or (b) issue the `SET RECOVERY SIMPLE` statement from a SQL query tool. Please see this reference for more details.

DB/VDB Memory Allocation

What is min/max memory in SQL Server?

Memory is allocated at the SQL Server level, so all the DBs will share the entire load. The max memory should be close to the maximum available on the server.

Why is it important to make this change?

To assure that masking jobs will perform at an optimum level.

How exactly do I make this change?

Use SQL Server Management Studio and change the max memory allocation for the server.

Primary/Foreign/DMS_ROW_ID Keys

What is a key?

A key is a unique, non-null value that identifies a row in the database.

Why is it important to make this change?

Using a PK or Foreign key is critical for fast updates. When a table does not have an identity column with an index or a PK/FK then the masking engine will alter the table to have an Identity column, DMS_ROW_ID to optimize performance.

How exactly do I make this change?

A logical key can be added to a table in the Masking Engine Ruleset for each table, if there is a specific column that would find the row to update faster than the current PK/FK.

Creating a Masking User and Privileges

It is highly recommended to create a database user, and possibly a role, for use by the Masking Engine. This user should be created in a non-Production environment and not in your production environment. The following permissions are needed:

- db_datareader
- db_datawriter
- db_ddladmin

SQL commands to add a user with the required privileges:

```
USE [mask_db]

GO

CREATE LOGIN [mask_user] WITH PASSWORD=N'delphix123'

GO

CREATE USER [mask_user] FOR LOGIN [mask_user]

GO

USE [mask_db]

GO

ALTER ROLE [db_datareader] ADD MEMBER [mask_user]

GO

USE [mask_db]

GO

ALTER ROLE [db_datawriter] ADD MEMBER [mask_user]

GO

USE [mask_db]

GO

ALTER ROLE [db_ddladmin] ADD MEMBER [mask_user]

GO
```

Preparing Sybase Database for Profiling and Masking

> ✏️ **Note**
>
> Masking large tables can result in large transactions (depending on the masking job's commit size). It is important to manage each database's transaction log as appropriate to allow the masking jobs to run. Failure to manage the transaction log can result in the suspension of the transaction and hence the masking job appears to hang. Please review the ASE documentation Managing Free Space with Thresholds on how to manage the transaction log threshold. Sometimes it is necessary to resize the database to have a larger transaction log. When resizing a Delphix VDB, take care to ensure that the any new log devices are created in the VDB's underlying "datafile" directory provided by the Delphix Engine. For more information please review **Resizing an SAP ASE VDB** located on https://docs.delphix.com/docs.

Before masking your data, it is important to prepare the database. This section explains the required changes, reasons for the change, and instructions to make the change.

What is min/max memory in SQL Server?

Determining the Amount of Memory SAP ASE Needs

The total memory SAP ASE requires to start is the sum of all memory configuration parameters plus the size of the procedure cache plus the size of the buffer cache, where the size of the procedure cache and the size of the buffer cache are expressed in round numbers rather than in percentages. The procedure cache size and buffer cache size do not depend on the total memory you configure. You can configure the procedure cache size and buffer cache size independently. Use **sp_cacheconfig** to obtain information such as the total size of each cache, the number of pools for each cache, the size of each pool, and so on.

Use **sp_configure** to determine the total amount of memory SAP ASE is using at a given moment:

```
1> sp_configure "total logical memory"
```

Parameter Name	Default	Memory Used	Config Value	Run Value	Unit	Type
total logical memory	33792	127550	63775	63775	memory pages(2k)	read-only

The value for the Memory Used column is represented in kilobytes, while the value for the Config Value column is represented in 2K pages.

The Config Value column indicates the total logical memory SAP ASE uses while it is running. The Run Value column shows the total logical memory being consumed by the current SAP ASE configuration. Your output differs when you run this command because no two SAP ASEs are configured exactly the same.

Determine the SAP ASE Memory Configuration

The total memory allocated during system start-up is the sum of memory required for all the configuration needs of SAP ASE. You can obtain this value from the read-only configuration parameter **total logical memory** . This value is calculated by SAP ASE. The configuration parameter **max memory** must be greater than or equal to **total logical memory**. **Max memory** indicates the amount of memory you will allow for SAP ASE needs.

During server start-up, by default, SAP ASE allocates memory based on the value of **total logical memory**. However, if the configuration parameter **allocate max shared memory** has been set, then the memory allocated will be based on the value of **max memory**. The configuration parameter **allocate max shared memory** enables a system administrator to allocate the maximum memory that is allowed to be used by SAP ASE, during server start-up.

The key points for memory configuration are:

- The system administrator should determine the size of shared memory available to SAP ASE and set **max memory** to this value.

- The configuration parameter **allocate max shared memory** can be turned on during start-up and runtime to allocate all the shared memory up to **max memory** with the least number of shared memory segments. A large number of shared memory segments have the disadvantage of some performance degradation on certain platforms. Check your operating system documentation to determine the optimal number of shared memory segments. Once a shared memory segment is allocated, it cannot be released until the server is restarted.

- The difference between **max memory** and **total logical** memory determines the amount of memory available for the procedure and statement caches, data caches, or other configuration parameters.

- The amount of memory SAP ASE allocates during start-up is determined by either **total logical memory** or **max memory**. If you set **alloc max shared memory** to 1, SAP ASE uses the value for **max memory**.

- If either **total logical memory** or **max memory** is too high:

  - SAP ASE may not start if the physical resources on your machine are not sufficient.

  - If it does start, the operating system page fault rates may rise significantly and the operating system may need to be reconfigured to compensate.

Why is it important to make this change?

To assure that masking jobs will perform at an optimum level.

Primary/Foreign/DMS_ROW_ID keys to for masking Sybase:

What is a key?

A key is a unique, non-null value that identifies a row in the database.

Why is it important to make this change?

Using a PK or Foreign key is critical for fast updates. When a table does not have an identity column with an index or a PK/FK then the masking engine will alter the table to have an Identity column, DMS_ROW_ID to optimize performance.

How exactly do I make this change? (exact commands, etc).

A logical key can be added to a table in the Masking Engine Ruleset for each table, if there is a specific column that would find the row to update faster than the current PK/FK.

Note Sybase ASE will create unavoidable log entries when a table is altered and will increase the log size significantly. If needed, run the masking jobs using the On-The-Fly method to avoid log file increases.

> ✏️ **Note**
>
> While performing a data copy, the database that contains the table must have select **into/bulkcopy/pllsort** turned on.

Creating a Masking User and Privileges:

It is highly recommended to create a database user, and possibly a role, to mask. This user should not be created in production but should be created in non-Production. The following permissions are needed:

Syntax to add user and give privileges:

```
sp_adduser mask_user;
```

```
CREATE user NEWUSER;
```

```
CREATE LOGIN mask_user WITH PASSWORD Delphix_123; --THIS MUST BE DONE IN MASTER
```

```
CREATE USER mask_user IDENTIFIED BY Delphix_123;
```

```
GRANT SELECT ON PII_V2 TO mask_user;  GRANT INSERT ON PII_V2 TO mask_user;  GRANT DELETE ON PII_V2 TO
mask_user;  GRANT ALTER ON PII_V2 TO mask_user;  GRANT UPDATE ON PII_V2 TO mask_user;
```

```
GRANT ALTER ANY TABLE TO mask_user;
```

Adaptive Server requires a two-step process to add a user: sp_addlogin followed by sp_adduser.

```
CREATE LOGIN MASK_SUPER_USER WITH PASSWORD Delphix_123;
```

```
sp_addlogin MASK_SUPER_USER, Delphix_123;
```

```
GRANT ROLE sa_role TO MASK_SUPER_USER;
```

Connecting Data

Managing Environments

This section describes how you can create and manage your environments in the masking service.

As a reminder, environments are used to group certain sets of objects within the Masking Engine. They can be thought of as folders/containers where a specified user can create manage connectors, rule sets, and jobs.

The Main Environment screen lists all the environments the logged in user has access to. It is the first screen that appears when a user logs into Delphix.

The main **environments** screen contains the following information and actions:

- **Environment ID** — The numeric ID of the environment used to refer to the environment from the Masking API.

- **Application** — A way to indicate the name of the application whose data will be managed within this environment.

- **Environment** — The name of the environment.

- **Purpose** — The purpose of the environment.

- **Jobs** — The number of jobs contained within the environment.

- **Edit** — Edit the environment. See more details below.

- **Export** — Export the environment. See more details below.

- **Copy** — Copy the environment. See more details below.

- **Delete** — Delete the environment. See more details below.

The environments on the screen can be sorted by the various informational fields by clicking on the respective field. In addition, the environments listed can be filtered using the **Search** field. See more details below.

Adding An Application

For an environment to be created, an application needs to be specified. Here are the steps to add an application:

1. On the main environments page, near the upper right-hand corner of the screen, click on the **Select Action** drop-down list and select the **Add Application** option.

2. The screen prompts you for the following items:

   a. Application Name

3. Click **Save** to return to the **Environments List/Summary** screen.
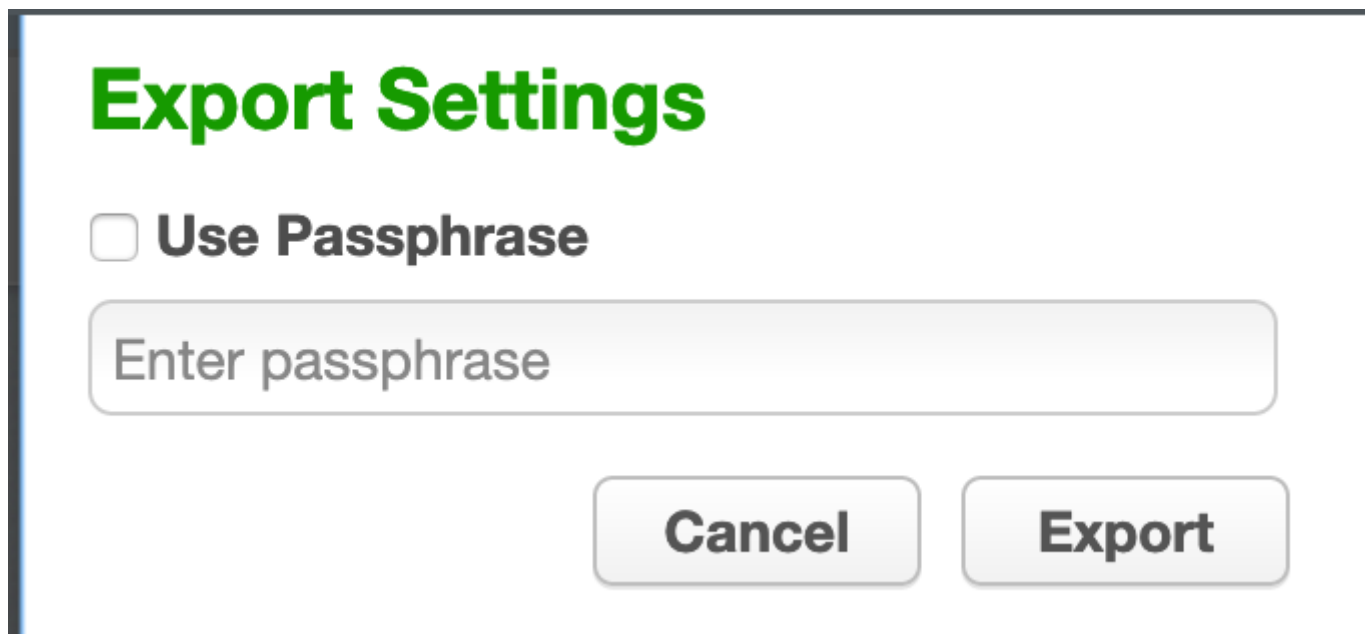
Creating An Environment

Here are the steps you need to take to create an environment:

1. On the main environments page, in the upper right-hand corner of the screen, click on the **Select Action** drop-down list and select the **Add Environment** option.

2. The screen prompts you for the following items:

3. **Application Name** – The name of the application to associate with the environment, for informational purposes.

4. **Environment Name** – The display name of the new environment.

5. **Purpose** – The type of masking workflow for the environment: Mask or Tokenize/Re-Identify.

6. **Enable Approval Workflow** – Whether or not to require approvals of inventories before masking jobs can be run in the environment.

7. Either click **Save** to return to the **Environments List/Summary** screen, or click **Save & View** to display the **Environment Overview** screen.

Exporting Settings

To export the Settings:

1. On the main environments page, in the upper right-hand corner of the screen, click on the **Select Action** drop-down list and select the **Export Settings** option.

2. The screen prompts you to take the input for the optional **Passphrase**. You can input the **Passphrase** by clicking the **Use Passphrase** checkbox.
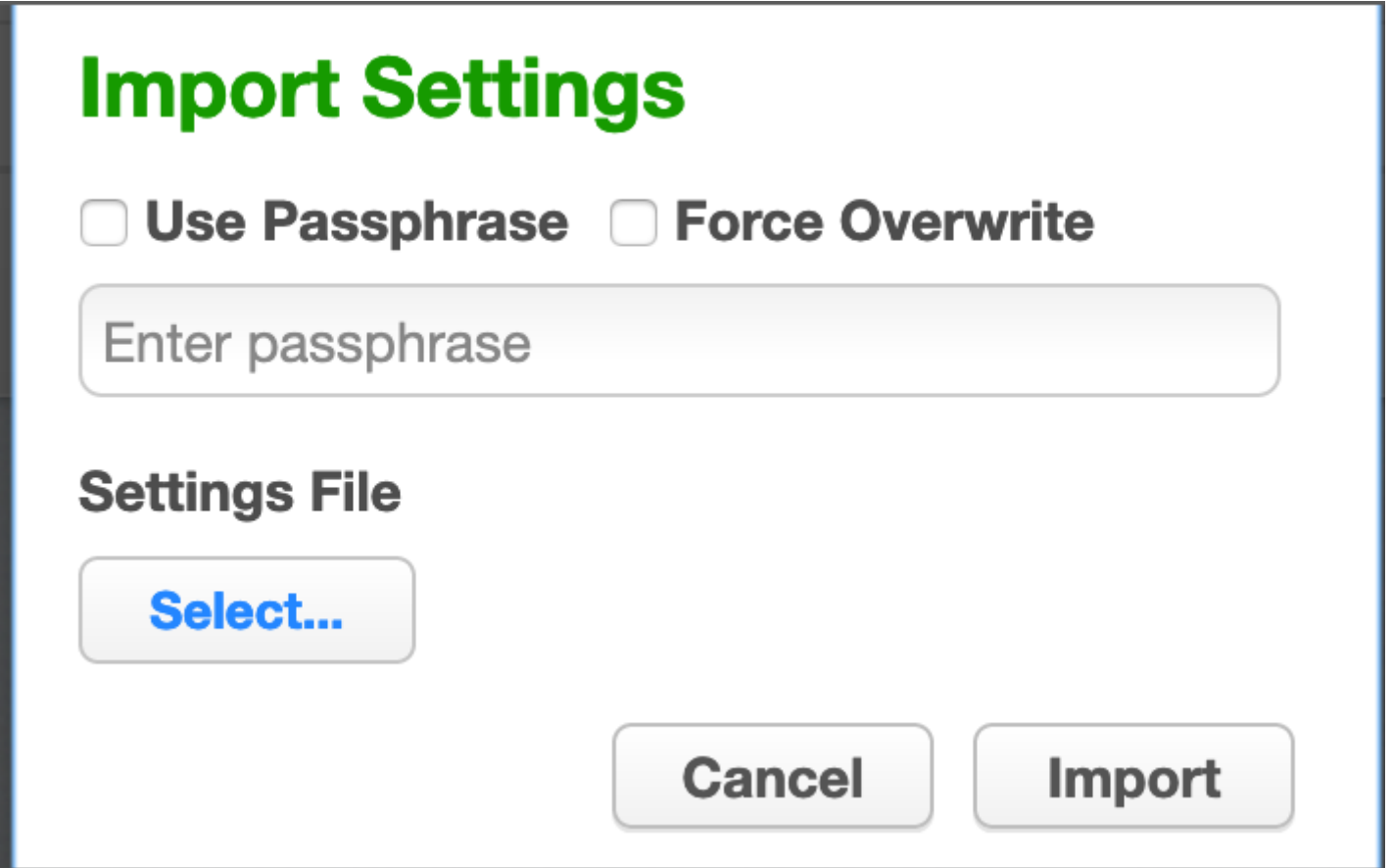
3. Click **Export**.

All the information related to Settings (Domain, Algorithm, File Format and so on) is exported to a file.

A status pop-up appears. You can wait to finish the download or you can close the download popup page to download the file for later. When the export operation is complete, automatically it will download the export file or you can click on the **Download file** name to download the export file manually. You can also check the export status from Async Task Status page.

Importing Settings

Once you have exported your settings, you can easily import it into another Masking Engine. To import settings:

Import Settings

☐ **Use Passphrase** ☐ **Force Overwrite**

Enter passphrase

Settings File

Select...

Cancel   Import

1. On the main environments page, in the upper right-hand corner of the screen, click on the **Select Action** drop-down list and select the **Import Settings** option.

2. The screen prompts you for the following items:

3. **Passphrase** – You can input the **Passphrase** by clicking the **Use Passphrase** checkbox. If the settings were exported using a passphrase then you must use the same passphrase for the import settings as well otherwise the import operation will fail.

4. **Force Overwrite** – Specify whether the import should fail if an object already exists with the same ID or the existing object should be overwritten. Click on the force overwrite checkbox if you want to overwrite the existing object.

5. **Settings File** – Click on **Select...** button to browse for the exported settings file that contains the information you want to import. (This file must be a previously exported masking environment.)

6. Click **Import** button to start the import operation.

A status pop-up appears. You can wait to finish the import operation or you can close the pop-up page and check the import status for later. When the import operation is complete, it will show the final status of the import operation on the pop-up page. You can also check the import status from Async Task Status page.

Async Task Status

To check the async task status:

1. On the main environments page, in the upper right-hand corner of the screen, click on the **Select Action** drop-down list and select the **Async Task Status** option.

2. A pop-up page will appear with the below filter options:

    a. **Select Task Type** : Select the type to filter the result.

    b. **Enter Async Task Id** : Enter the Async Task Id to filter the result.

3. Click on **Find** button to find the async task.

From the result grid, you can also download the export file for the export operation by clicking the **Download file** link on the corresponding row. You can also download the log file for the failed import/export operations by clicking the **Download log file** link on the corresponding row

Exporting An Environment

For a variety of different reasons (the main one being moving environments between masking engines), you may want to export all the objects within an environment (connectors, rule sets, masking jobs, etc).

To export an environment use the Export Environment option available in the Masking UI. To export an individual environment:

1. Click the **Export** icon or click on **Export** button on the **Environment Overview** screen.

2. The pop-up fills in the following items:

   a. Environment Name

3. You can input the optional **Passphrase** by clicking the **Use Passphrase** checkbox.

4. Click **Export**.

All the information for the specified environment (connectors, rule sets, inventory, jobs, and so on) is exported to a file.

A status pop-up appears. You can wait to finish the download or you can close the download pop-up page to download the file for later. When the export operation is complete, automatically it will download the export file or you can click on the **Download file** name to download the export file manually. You can also check the export status from Async Task Status page.

Importing An Environment

Once you have exported your environment, you can easily import it into another Masking Engine. To import an environment:

Import Environment

☐ **Import Settings** ☐ **Force Overwrite**

Application

◯ Existing ⦿ New

> Enter New Application Name

Environment

◯ Existing ⦿ New

> Enter New Environment Name

☐ **OTF Environment**

☐ **Use Passphrase**

> Enter passphrase

Environment File

> Select...

> Cancel   Import

1. On the main environments page, in the upper right-hand corner of the screen, click on the **Select Action** drop-down list and select the **Import Environment** option.

2. The screen prompts you for the following items:

3. **Import Settings** – Click the checkbox if you want to import settings as well.

4. **Force Overwrite** – Specify whether the import should fail if an object already exists with the same ID or the existing object should be overwritten. Click on force overwrite checkbox if you want to overwrite the existing object.

5. **Application** – You can select the existing application from the application drop-down or you can enter the application name to create a new application.

6. **Environment** – You can select the existing environment from the environment drop-down or you can enter the environment name to create a new environment.

7. **OTF Environment** – Click on **OTF Environment** checkbox to import the on-the-fly connectors into that environment. You can select the existing environment from the environment drop-down or you can enter the environment name to create a new environment.

8. **Passphrase** – You can input the **Passphrase** by clicking the **Use Passphrase** checkbox. If the exported file is used the passphrase then you should use the same passphrase for the import as well.

9. **Settings File** – Click on **Select...** button to browse for the exported settings file that contains the information you want to import. (This file must be a previously exported masking environment.)

10. **Environment File** – Click on **Select...** button to browse for the exported environment file that contains the information you want to import. (This file must be a previously exported Masking environment.)

11. Click **Import** button to start the import operation.

A status pop-up appears. You can wait to finish the import operation or you can close the popup page and check the import status for later. When the import operation is complete, it will show the final status of the import operation on the pop-up page. You can also check the import status from Async Task Status page.

Editing An Environment

To change the properties of an environment, do the following:

1. Click the **Edit** icon to the right of the environment status.

2. The pop-up prompts you for the following information:

   a. Environment Name

   b. Purpose

   c. Application Name

   d. Enable Approval Workflow

3. Click **Save**.

Copying An Environment

A user can also easily create an exact copy of a certain environment. This is a very powerful feature when wanting to have several similar but not exact environments but don't want to start from scratch. To copy an environment do the following:

1. Click the **Copy** icon to the right of the environment status.

2. The pop-up prompts you for the following information:

   a. Environment Name

    b. Purpose

    c. Application Name

    d. Enable Approval Workflow

3. Click **Save**.

Deleting An Environments

To delete an environment:

- Click the **Delete** icon to the right of the environment status and copy icon.

> ⚠️ **Warning**
>
> Clicking the **Delete** icon deletes EVERYTHING for that environment: connections, inventory, rule sets, and so on. It does not delete universal settings like algorithms, domains, etc.

Searching For Environments

When a large number of environments have been created on a Masking Engine, it may be useful to filter the **Environments List/Summary** screen. To filter the environment list, do the following:

1. In the **Search** field in the upper left side of the screen, enter the characters to search by.
2. Click the adjacent **Search** button.
3. The screen will display only the environments whose name match the specified search characters.

To re-display, the entire list of environments, clear the **Search** field of characters and click the **Search** button again.

Managing Remote Mounts

This section describes how you can mount a NFS/CIFS location inside the Masking Engine and use it in a masking job.

In order to access the files shared over NFS/CIFS server from the Masking Engine, complete the following two steps:

1. Create and connect a mount using Mount Filesystem API endpoint.

2. Create a file connector with Filesystem Mount Point mode. Or, Upload a XML/Copybook file format using Filesystem Mount Point mode.

Mount Filesystem API

The **Mount Filesystem** APIs are used to perform normal CRUD operations(Create, Read, Update and Delete) along with three mount operations connect(mount), disconnect(unmount) and remount on a mount object.

Mount Information

To create a mount entry, information about the mount are passed. Some of them are required and some are optional.

- Require Information:

  - *mountName*: The name of the mount. This name is used to refer this mount in the connector creation and file format upload UIs.

  - *hostAddress*: The NFS/CIFS server address.

  - *mountPath*: The remote path shared by the NFS/CIFS servers.

  - *type*: The type of the server. CIFS, NFS3, or NFS4.

- Optional Information:

  - *options*: The mount options.

  - *connectOnStartup*: Whether this mount should be connected or not when the server starts.

> ✏️ **Note**
>
> When a server shuts down, all the mounts are disconnected.

Mount Options

The API supports passing many mount options. Not all of them are supported by a server. After a mount is connected, you might see the options field has many options that were not passed by you or some options have been eliminated that were passed by you. The options field shows effective options only. The applied options are gathered after a mount is connected.

The API also restricts usage of some mount options.

Enforced Options

The following mount options are enforced and added to the list of options for all mounts:

- *nosuid*: The filesystem cannot contain set userid files.
- *noexec*: No executable script can be run from the mount.
- *nodev*: The filesystem cannot contain special devices.

Minimal Options

Although `options` is an optional field, it is required for CIFS mounts to pass credentials. the following options are required for CIFS mounts:

- *username*: The username to connect to the CIFS server.
- *password*: The password of the user.
- *domain*: The domain of the user.

For example, `"options": "username=abc,password=pass,domain=DOMAIN"`

For NFSv3 mounts, `options` are not required, therefore can be `null`.

For NFSv4 mounts, the following option is required:

- *nfsvers*: The NFS protocol version number. For example, "options": "nfsvers=4.0"

Version Options

The version information is passed using *vers* option. The supported versions based on mount types are

Mount Type	Supported Versions
CIFS	2.0, 2.1, 3.0
NFS3	3, 3.0
NFS4	4, 4.0, 4.1, 4.2

Generic Options

Some mount options are generic which can be applied to all the mount types while some are mount specific options. In the case of *remount* operation, only generic options can be modified. The list of allowed generic options are:

async, atime, auto, context, defaults, defcontext, diratime, dirsync, fscontext, group, iversion, lazytime, loud, mand, _netdev, noatime, noauto, nodev, nodiratime, noexec, nofail, noiversion, nolazytime, nomand, norelatime, nostrictatime, nosuid, nouser, owner, relatime, _rnetdev, ro, rootcontext, rw, silent, strictatime, sync, and user.

CRUD Operations

Create

The *create* endpoint is used to create a mount entry. It takes all the information about a mount as its input and creates a mount entry. It doesn't do any kind of validation about the mount's accessibility. The validation is done during the *connect* operation.

Read

The *read* endpoints are used to retrieve information about a mount. There are two *read* endpoints. 1. *get all*: To get information about all mounts. 2. *get*: To get information about any particular mount identified by its id.

Update

The *update* endpoint is used to modify any information of a mount. Update operation can be performed only on a disconnected mount.

Delete

The *delete* endpoint is used to delete a mount entry. A mount can be deleted only if it is not being used in any of the connectors.

Mount Operations

Apart from normal CRUD operations, there are three special mount related operations exposed through the API.

Connect

The *connect* endpoint is used to mount a remote mount inside the masking engine. If the connect operation succeeds then, the options field is updated with the applied mount options.

Disconnect

The *disconnect* endpoint is used to unmount a remote mount from the Masking Engine.

Remount

The API supports the **remount** operation. This can be used to remount an active or to connect a disconnected mount and also to update some mount information. This can update *mountName*, *connectOnStartup* and generic *options* only. For other updates, use the normal update API.

Resolve Mount Consistency

A script runs in the background to keep the data in the mount_information table and mounts in sync. If for some reason, the data for a mount mounted inside the mount engine and data corresponding to that mount in mount_information table becomes inconsistent, the mount is unmounted. For example, if a mount is in a disconnected state in DB but it is mounted in the engine, then it will be unmounted.

Using Mounts

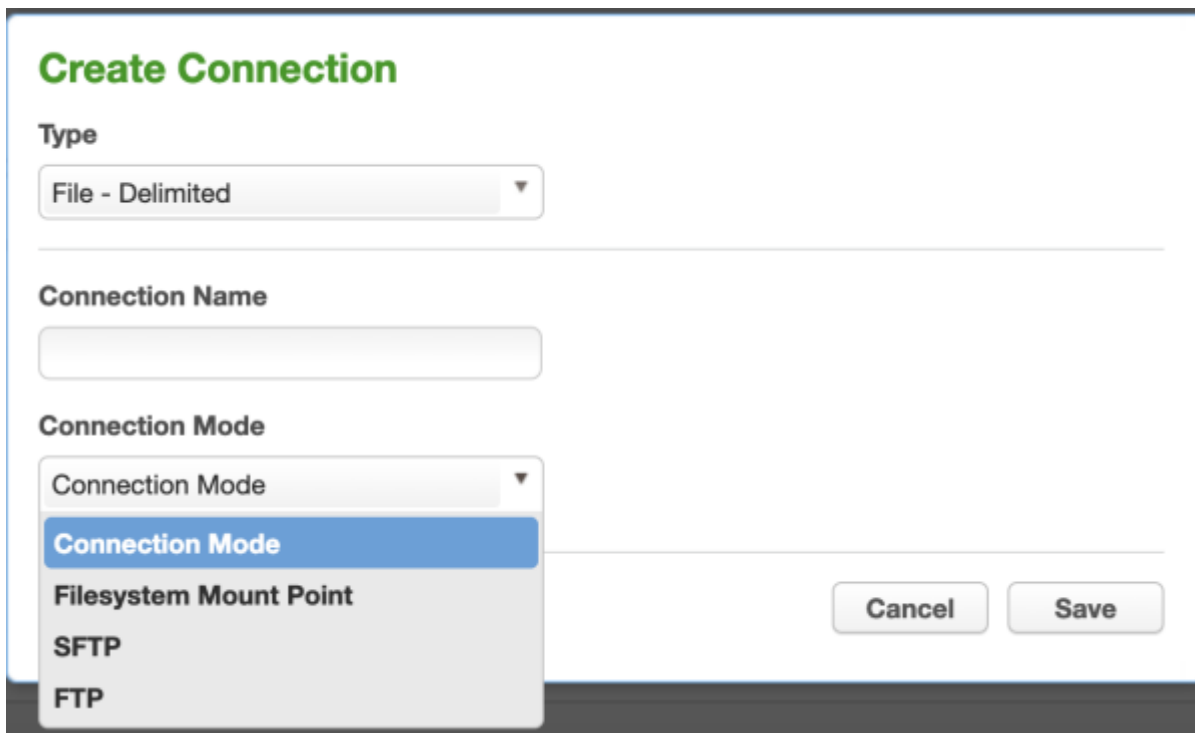A mount can be used at two places:

- File connectors
- File formats

File Connector

While creating a connector, when any file connector option is selected, the UI shows a dropdown to select how a file will be accessed. There are three options:

- Filesystem Mount Point
- SFTP
- FTP

On selecting the Filesystem Mount Point option, the mount name and a path inside the mount needs to be specified.

- Mount Name: This is a list of mount names created in the engine.

- Path Under Mount: A path relative to the path mounted. By default, it is at the root of the remote Mount path.

- Remote Path: The complete remote path. On selecting a mount name and typing a path in the above input box, this gets updated.

> ✏️ **Note**
>
> A connector can be created even if a mount is in a disconnected state but it should be in an active state when a ruleset is being created or when a job is run.

File Format

The XML and Copybook file formats can be uploaded from a remote location. To upload a file format from an NFS/CIFS location, select the Filesystem Mount Point option.

Sync Mounts

A mount can be synced from a source engine to a target engine using Sync APIs. Syncing a file connector using a mount also syncs the related mounts. The following mount information fields are synced:

- mountName
- hostAddress

- mountPath

- options

- connectOnStartup

- type

In case of CIFS mounts, the password is not synced. In order to set the password in the target engine, update the mount's options and ensure to include the password in the options.

Recommended Mount Server Configuration

The NFS and CIFS servers should be configured in such a way that the files are readable and writeable by the Masking Engine.

CIFS Server

The user-provided to connect to the mount should have read and write permission on the mount.

NFS Server

1. The Masking Engine's server IP should have read and write permission on the mount.

2. For NFS, the access to a file is controlled based on the UID and GID. In order to give read & write permission to the Masking Engine on the share path, the path should be shared with the following options:

```
<mount path> <masking engine ip>(rw,all_squash,anonuid=<uid>,anongid=<gid>)
# uid and gid is of the owner of the shared path on the server
```

Managing Connectors

This section describes how you can create and manage your connectors.

As a reminder, connectors are the way users define the data sources to which the Masking Engine should connect. Connectors are grouped within environments. In order to navigate to the **connectors** screen, click on an environment and then click the **Connector** tab.

The **connectors** screen contains the following information and actions:

- **Connector ID** — The numeric ID of the connector used to refer to the connector from the Masking API.

- **Connector** — The name of the connector.

- **Meta Data Source** — The type of connector. One of Database, File, or Mainframe.

- **Type** — The specific type of connector.

- **Edit** — Edit the connector. See more details below.

- **Delete** — Delete the connector. See more details below.

The connectors on the screen can be sorted by the various informational fields by clicking on the respective field.

Creating a Connector

To create a new connector:

1. In the upper right-hand corner of the **Connector** tab, click **Create Connection**. The **Create Connection** window appears, prompting you for connection information for the data source you would like to connect to. The required information will change depending on the **Type** of data source you select. For more details on what info is needed to connect to different types (Oracle, AWS RDS, etc) see sections below.

2. Several of our connector types offer two different modes of connecting, **Basic** and **Advanced Mode**. Advanced Mode gives you the ability to specify the exact JDBC URL and add parameters that may not be available in Basic Mode.

Create Connection

Type

Database - Oracle ▾ ● Basic ○ Advanced

Connection Name	**Port**
Flash	1521

Schema Name ☑ Use Kerberos Authentication

FLASHXDB

Host Name/ IP **Principal Name**

10.1.0.20

SID **Password**

XEXE LEAVE BLANK TO USE KEYTAB

[Test Connection] [Cancel] [Save]

The fields that appear on the Connector screen are specific to the selected Connector Type (see Connector Types below).

3. Click **Save**.

Editing a Connector

To edit a connector:

1. In the **Connector** tab, click the **Edit** icon for the connector you want to edit.

2. Change any information necessary. To change the password:

    a. Select the checkbox next to **Change Password**.

    b. In the field that appears, enter the new **password**.

3. Click **Save**.

Deleting a Connector

To delete a connector, click the **Delete** icon to the far right of the connector name.

Warning: When you delete a connector, you also delete its rule sets and inventory data.

Connector Types

Database Connectors

The fields that appear are specific to the DBMS Type you select. If you need assistance determining these values, please contact your database administrator.

You can only create connectors for the databases and/or files listed. If your database or file type is not listed here, you cannot create a connector for it.

- **Connection Type** — (Oracle, MS SQL Server, and Sybase only) Choose a connection type:

  - **Basic** — Basic connection information.
  - **Advanced** — The full JDBC connect string including any database parameters.

- **Connection Name** — The name of the database connector (specific for your Delphix application).

- **Schema Name** — The schema that contains the tables that this connector will access.

- **Database Name** — The name of the database to which you are connecting.

  Note: The database name field is case-sensitive. It must match exactly with the name of the current database as known to the instance.

- **Host Name/ IP** — The network hostname or IP address of the database server.

- **Use Kerberos Authentication** - (Oracle only, optional) Whether to use Kerberos to authenticate to the database. This box is clear by default. Before Kerberos may be used, the appliance must be properly configured - refer to these instructions (link to appliance Kerberos configuration instructions[1]). If this box is checked, the application authenticates with the Kerberos KDC before connecting to the database, then uses its Kerberos credentials to authenticate to the database instead of a login/password. When Kerberos is enabled, the "Login ID" field is treated as the Kerberos user principal name. The password, if supplied, is used to authenticate the user principal with the KDC. The password field may be left blank if the keytab set during appliance configuration contains keys for the user principal.

- **Login ID** — The user login this connector will use to connect to the database (not applicable for Kerberos Authentication).

- **Password** — The password associated with the Login ID or Username. (This password is stored encrypted.)

- **Principal Name** - (Kerberos Authentication only) The name of the Kerberos user principal to use when authenticating with the KDC. The realm portion of the principal may be omitted if it matches the configured default realm.

- **Service Principal** - (Sybase with Use Kerberos Authentication only) The name of the Sybase service instance.

- **Port** — The TCP port of the server.

- **SID** — (Oracle only) Oracle System ID (SID).

- **Instance Name** — (MS SQL Server only) The name of the instance. This is optional. If the instance name is specified, the connector ignores the specified "Port" and attempts to connect to the "SQL Server Browser Service" on port 1434 to retrieve the connection information for the SQL Server instance. If the instance name is provided, be sure to make exceptions in the firewall for port 1434 as well as the particular port that the SQL Server instance listens to.

- **Custom Driver Name** — (Generic only) The name of the JDBC driver class, including Java package name.

- **JDBC URL** — (Generic and Advanced connector mode for Oracle, MS SQL Server, and Sybase only) The custom JDBC URL, typically including hostname/IP and port number.

- **Connection Properties File** - A Java properties file to specify configurations for the JDBC connection. See Database Connection Properties for more information.

All database types have a **Test Connection** button at the bottom left of the New Connector window. We highly recommend that you test your connection before you save it. Do so before you leave this window. When you click **Test Connection**, Delphix uses the information in the form to attempt a database connection. When finished, a status message appears indicating success or failure.

File Connectors

The following values appear when any of the file connector types are selected:

- **Connector Name** — The name of the file connector (specific to your Delphix application and unrelated to the file itself).
- **Connection Mode** — Filesystem Mount Point, SFTP and FTP

The rest of the values appear based on the selected **Connection Mode** value. For **Filesystem Mount Point** connection mode, refer to the corresponding section in the Managing Remote Mounts page. For other connection modes, the following values appear:

- **Path** — The path to the directory where the file(s) are located.
- **Server Name** — The name of the server used to connect to the file.
- **Port** — The port used to connect to the server.
- **User Name** — The user name to connect to the server.
- **Password** — (non-Public Key Authentication only) The associated password for the server.
- **Public Key Authentication** — (Optional) (Only appears for SFTP.) Check this box to specify a public key. When you check this box, the **Available Keys** drop-down appears. Choose a key from the drop-down. See Delphix Masking APIs for information on uploading public keys to the Masking Engine.

  Note: If you plan to do on-the-fly masking then you will need to create a separate environment and connector to be the source for the files to be masked. The masked files will get put into the directory pointed to by the connector you created previously (the target). However, the file path specified in the connector of the target rule set must point to an existing file the target directory. It does not have to be a copy of the file, just an entry in the directory with the same name. It will be replaced by the masked file.

Starting version 6.0.9.0 the SFTP mode is extended with the 'User Directory as root' flag. If the Path defined is relative to the User-home-dir as configured on the SFTP Server, tick the flag below.

If connector is configured via the API than that flag is accessible as "userDirIsRoot", for example:

```
{
    "connectorName": "Test SFTP Connector",
    "environmentId": 2,
    "fileType": "DELIMITED",
    "connectionInfo": {
        "connectionMode": "SFTP",
        "path": "/delimited",
        "host": "yourSFTPServer",
        "loginName": "xxxxx",
        "password": "xxxxx",
        "port": 22,
        "userDirIsRoot": true
    }
}
```

Database Connection Properties

Getting Properties

To retrieve all properties set on the connector, make a request to the `GET database-connector/{id}/properties` endpoint. This endpoint will respond with all default properties set by the driver, superimposed by any properties specified by an uploaded connection properties file. If a properties file is uploaded for a connector, this list can also be viewed through the UI on the database connector form, where you can sort by `Property`, `Value`, or `Modified`. The `Modified` field signifies whether the property value is the default or modified by the uploaded properties file.

> ✏️ **Note**
>
> The database name field is case-sensitive. It must match exactly with the name of the current database as known to the instance.

> ✏️ **Note**
>
> Only a valid JDBC URL is required to retrieve properties of a connector; a valid connection to the database server is not necessarily required.

Edit Connector: mssql-test

Database - mssql

⦿ Basic ○ Advanced

Connection Name

mssql-test

Schema Name

schema

Database Name

db

Host Name/ IP

mssql-server.test.co

Instance Name

MSSQLSERVER

Port

1433

☐ Use Kerberos Authentication

Login ID

user

☐ Change Password

Custom Properties File ❓

Select...   View

mssql_properties.properties ⊗

Test Connection   Cancel   Save

Example of properties on the built-in MSSQL database connector

Properties

Property	Value	Modified
keyVaultProviderClientKey		false
cancelQueryTimeout	-1	false
workstationID		false
fips	false	false
encrypt	false	false
selectMethod	direct	false
sslProtocol	TLS	false
jaasConfigurationName	SQLJDBCDriver	false
trustManagerClass		false
socketFactoryClass		false
serverSpn		false
clientCertificate		false
columnEncryptionSetting	Disabled	false
multiSubnetFailover	false	false
applicationIntent	readwrite	false
serverNameAsACE	false	false

OK

Setting Properties

Properties can sometimes be set through the JDBC URL or through a connection properties file. Customizing the JDBC URL is limited to Advanced, Generic, and Extended Connectors, while uploading a properties file is supported by all database connectors. All properties files must have the extension `.properties` and must adhere to Java properties file syntax. Even if a property specified in the properties file is not technically supported by the JDBC driver, it will still be passed along to the driver when building the JDBC Connection. All provided and unsupported properties will be logged whenever the properties file is loaded.

> ✏️ **Note**
>
> The properties file is assumed to be written using ISO 8859-1 character encoding.

```
3. vi mssql_properties.properties (vim)
1 queryTimeout=1000
2 portNumber=1433
3 databaseName=test_db

-- INSERT --
```

> ✏️ **Note**
>
> Certain JDBC drivers do not allow specific supported properties to be set through the JDBC URL. For example, Caché DB does not support setting the schema property through the JDBC URL, so it must either be set through the extended connector's Schema form field or through a connection properties file. Please defer to specific Driver documentation to see how a property must be set.

> ✏️ **Note**
>
> When a property can be duplicated among a form field, the JDBC URL, and the properties file, the property value will most likely be used in the following hierarchy of specification.
>
> 1. Connector form fields (where applicable) for username, password, and schema
>
> 2. Properties file 3a. Connector form fields (where applicable) for database name, host, port, SID, and instance name 3b. JDBC URL
>
> Though this hierarchy is convention, it is up to the JDBC driver to implement the precendence for duplicate properties specified among the URL, Properties object, and JDBC Connection API. Please defer to the specific JDBC Driver documentation to verify which method of specification precedes the other. A Delphix Masking connectors form will either have the fields listed in 3a or 3b, but not both. Therefore, it is not possible to duplicate a property between 3a and 3b.

Security Considerations

The property key or value provided in a database connector's properties file will not be regulated and is subject to any user with `CREATE or UPDATE connector` privileges. **This means that even supported sensitive properties such as `user`, `password`, `hostname`, etc... will be available in plain text to anyone with the `VIEW connector` privilege.**

If possible, specify sensitive properties through relevant form fields which will be obfuscated in all places or through the JDBC URL which will still be visible in plain text to any user with the `VIEW connector` privilege but will be redacted in support bundles.

Managing Extended Connectors

Extended Connectors allow you to upload additional JDBC Drivers to the Delphix Masking Engine to enable masking of data sources not natively supported by Delphix Masking.

Limitations

Delphix supports type 4 JDBC Drivers. These must be a pure-java .jar file that can be used simply by uploading it (or it's zip file) to the engine. Anything that requires compilation on the engine, or execution of any kind of install or licensing script, is not supported.

Extended Connectors don't support all of the features available for built-in connectors like Oracle. As of 6.0.9.0, the "Disable Constraint", "Disable Trigger" and "Drop Indexes" options can be implemented and enabled by driver support plugins, which are detailed here. Delphix provides support for Extended Connectors in accordance with our Support Policy

Drivers that require a Java version higher than 8 are not supported.

Installing a new driver

To use a new JDBC driver, first you need to upload it to your Masking Engine. Since some drivers require multiple files, the driver and any additional files it needs to function should be put together in a single zip file. Even if a driver doesn't require additional files, it still needs to be zipped.

For example, to package the Informix JDBC driver for use with Delphix Masking take all three files provided for Informix and zip them together:

```
$ ls
LICENSE.txt ifxjdbc.jar ifxlang.jar
$ zip informix.zip *
  adding: LICENSE.txt (deflated 70%)
  adding: ifxjdbc.jar (deflated 4%)
  adding: ifxlang.jar (deflated 4%)
$ ls
LICENSE.txt ifxjdbc.jar ifxlang.jar informix.zip
$
```

To upload the driver package to the engine, navigate to the **JDBC Drivers** under **Settings**
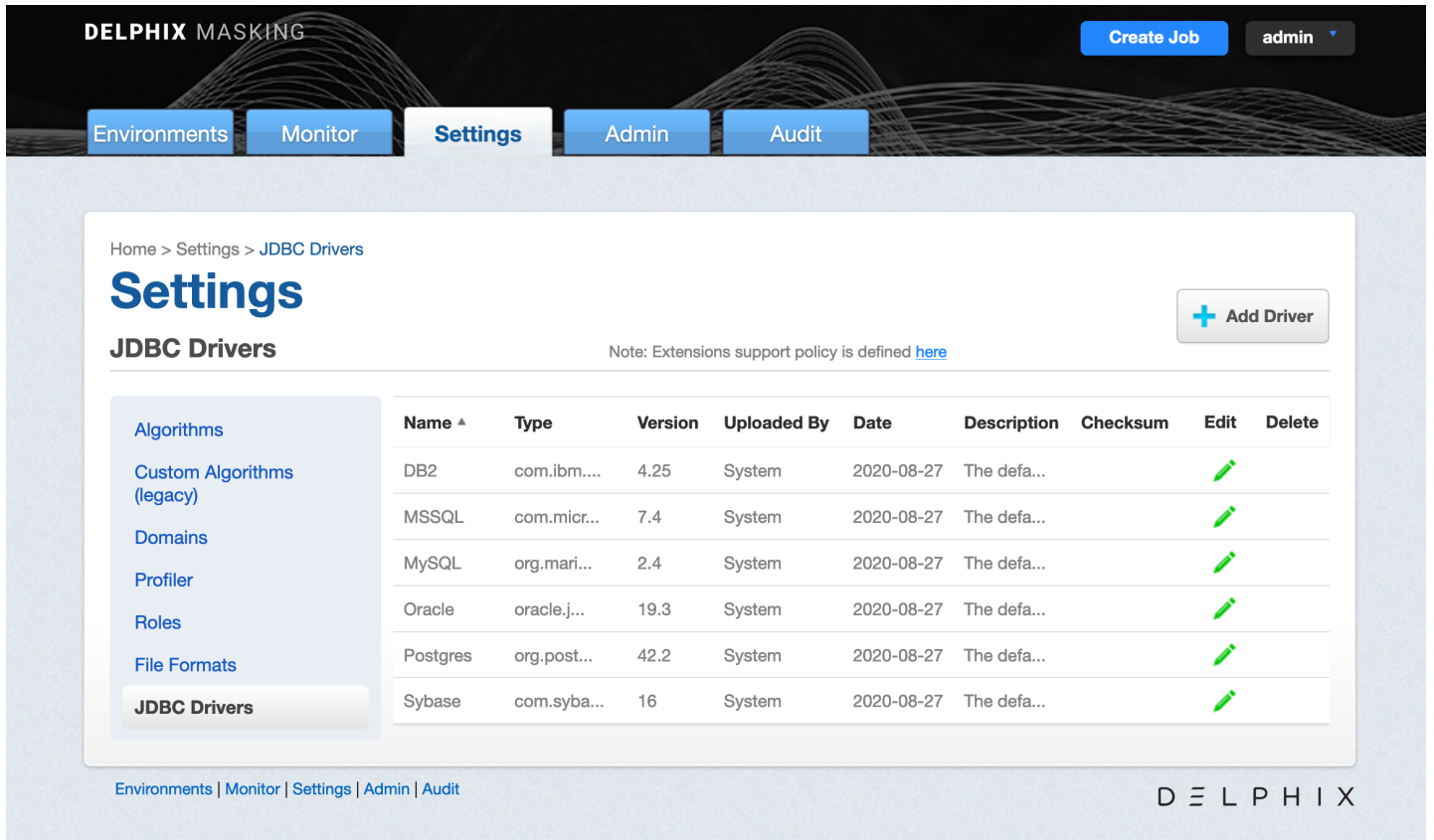
Clicking **Add Driver** will bring up a dialog box to upload the driver zip file and enter the driver's configuration details.
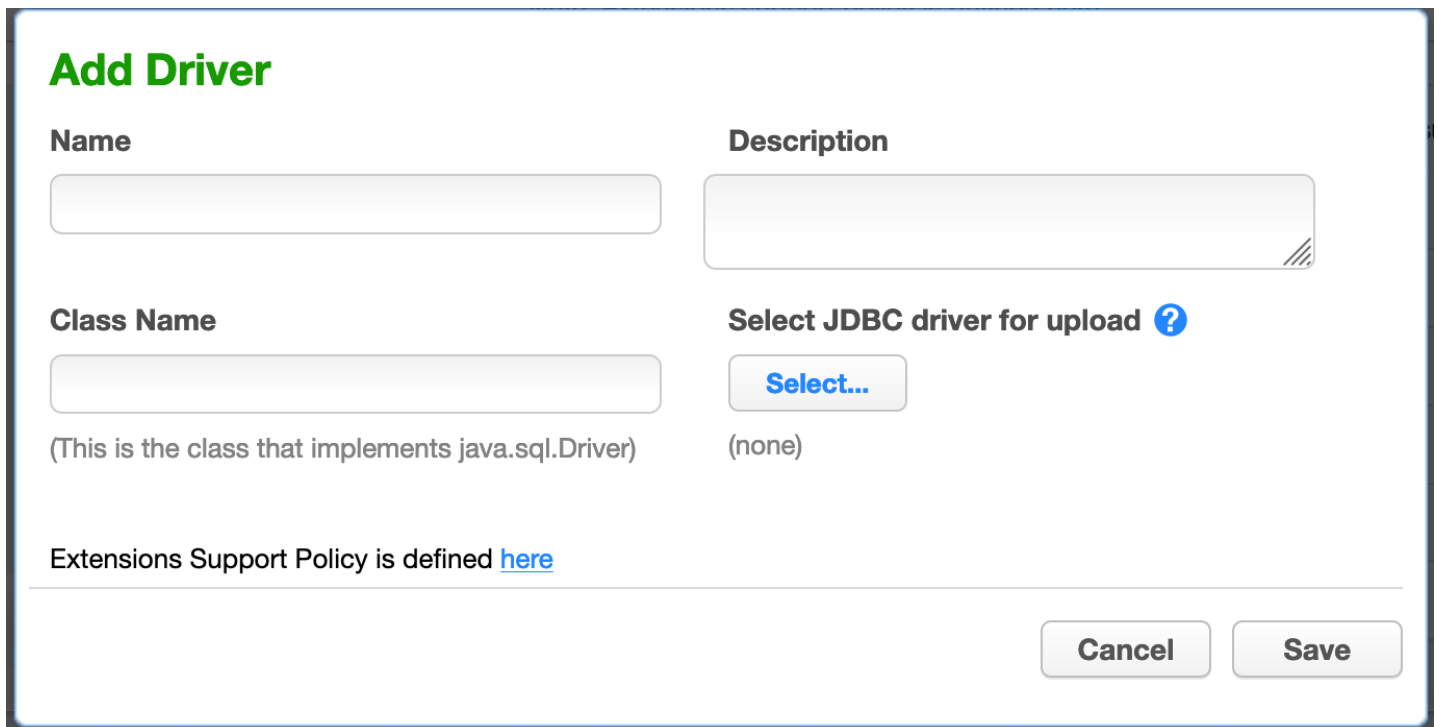
The **Add Driver** screen lets you set the following information.

- **Name** A human-readable name for the driver. Name it whatever is convenient for you. **Note:** Special Characters are not allowed in the Name field.
- **Description** A human-readable description of the driver.

- **Class Name** The Fully Qualified Class Name of the class in the JDBC driver that implements the java.sql.Driver interface. The class name will be in the documentation for the driver itself.
- **Select JDBC driver for upload** Lets you select the zip file containing the driver and upload it.
- **Permissions** Additional permissions required by the driver. See the Driver Permissions section

> ✏️ **Note**
>
> Users cannot update the driver support that a jdbc driver references or uses via the UI; as of 6.0.9.0, that can only be done via the web API.

To remove an uploaded driver, click the **Delete** button to the right of the connector on the **JDBC Drivers** tab. Note that the delete will fail if any Connectors exist that use the driver you're trying to delete.

If you find you need to edit a driver's configuration options later, click the pencil icon next to the driver's listing on the **JDBC Drivers** tab.

Driver Permissions

The Delphix Masking Engine uses the Java Security Manager to prevent uploaded JDBC drivers from performing certain actions without your permission.

Uploaded drivers are granted all permissions *except* for the following non- `FilePermission` :

Class	Target	Action
java.net.SocketPermission	localhost:-	accept, connect, listen, resolve
java.lang.RuntimePermission	exitVM	
java.lang.RuntimePermission	createClassLoader	
java.lang.RuntimePermission	accessClassInPackage.sun	
java.lang.RuntimePermission	setSecurityManager	
java.security.SecurityPermission	setPolicy	
java.security.SecurityPermission	setProperty.package.access	

With regards to FilePermissions, `read` access is granted to all, though `write` is only allowed for the following directories:

- the masking user's home directory (`System.getProperty("user.home")`)

- the JVM's default temp directory (`System.getProperty("java.io.tmpdir")`)

Please note that both of these locations are shared, so care will need to be taken to avoid collisions.

The set of permissions granted to uploaded drivers is static and cannot be modified.

Extended Logging

The Delphix Masking Engine provides enhanced logging for extended connectors to assist in debugging connection problems. Enhanced logging can be enabled when the connector is created by checking the 'Enable Logger' box. Enhanced logging may have an impact on performance so you should enable it only when debugging connection problems.

Note that extended logging will not work with signed drivers such as MSSQL.

Enhanced Logging requires some additional permissions to be granted.

Class Name	Target Name	Action Name	Purpose
java.io.RuntimePermission	getClassLoader		Allows the driver to load the classes implementing the logging feature

Creating an Extended Connector

Creating a connector using an Extended Driver is very similar to creating a connector with built-in support. Choose **Database - Extended** as the Type. The following fields will be available:

- **Connection Name** A name for this connection
- **JDBC Driver** Select the JDBC Driver you want to use for this connection
- **Login ID** The username the Masking Engine should connect to the target database with.
- **Password** The password to use to connect to the database
- **JDBC URL** You must provide the JDBC URL for the database to connect to. The exact format and available parameters are specific to the database you're connecting to. Consult your database vendors documentation for details.

> ✏️ **Note**
>
> Some databases allow you to specify usernames and passwords in the JDBC URL. It's best not to do this. The Delphix Masking Engine is careful not to log the Login ID and Password in the Masking Engine's logs, but JDBC URLs may be logged unmodified.

Once the connector is created, you can create rulesets, inventories, and jobs to profile and mask your data as with other types of connectors.

Extended Connectors can be edited and deleted in the same way as Built In Connectors

Synchronization

Connectors using extended JDBC Drivers can be synchronized similarly to other connectors. See Working with Multiple Masking Engines for details. When a job or connector requires an uploaded JDBC Driver, the driver will be exported along with the connector or job. JDBC Drivers are part of the **Global Object** and so will be synchronized whenever the Global Object is synchronized. They can also be synchronized individually.

Managing Rule Sets

This section describes how Rule Sets can be created, edited, and removed.

The Rule Sets Screen

From anywhere within an Environment, click the **Rule Set** tab to display the Rule Sets associated with that environment. The **Rule Sets** screen appears. If you have not yet created any rule sets, the Rule Set list is empty.

The **Rule Sets** screen contains the following information and actions:

- **Rule Set ID** — The numeric ID of the rule set used to refer to the rule set from the Masking API.

- **Name** — The name of the rule set.

- **Meta Data Source** — The type of rule set. One of Database, File, or Mainframe.

- **Type** — The specific type of ruleset.

- **Edit** — Edit the rule set. See more details below.

- **Refresh/Save** — Refresh the rule set. Only applies to Database rule sets. See more details below.

- **Copy** — Copy the rule set. See more details below.

- **Delete** — Delete the rule set. See more details below.

The rule sets on the screen can be sorted by the various informational fields by clicking on the respective field.

The Create/Edit Rule Set Window

In the upper right-hand corner, click the **Create Rule Set** button.

The **Create Rule Set** window appears.

1	**Rule Set Name Input Field**
	When editing an existing rule set, this field will be filled with the existing rule set name by default.
2	**Connector List**
	When creating a new rule set, all available connectors will be listed here. When editing an existing rule set, only the connector currently in use will appear.
3	**Table or File List**
	If a database connector is selected in the connector list, all available tables in the database schema associated with the connector will appear in this list. If a file connector is selected, all available files in the directory associated with the connector will appear in this list.
4	**Selected Table or File Number**
	Displays how many tables or files you have selected.
5	**Search Query Input Field**
	You can enter a search query here. After typing the search query, press **ENTER** to execute the search query.

✏️ **NOTE - search query**

- Use * to match any characters in the names of tables or files.

- If you have selected a table or file before searching and it is not in the search results, it will not be included in the rule set. You can add back the table or file by removing the search query.

- Checkbox / selections do not persist through a search or a clearing of the search field.

6	**Clear Search Button**
	Click to remove any search query.

7	**Select All Button**
	Click to select all tables or files in the table or file list.

8	**Clear All Button**
	Click to deselect all tables or files in the table or file list.

9	**File Name Patterns Editor**
	This editor will appear only when the selected connector is a file connector.

10	**Add File Pattern Button**
	Click to add a new file pattern entry below.

11	**File Pattern Input Field**
	Enter the file pattern here.

✏️ **NOTE - file pattern syntax**

- Expressions are case sensitive.

- A file pattern uses the regular expression syntax defined by the Java Pattern class. The syntax is documented here.

- For example, the pattern

  `.*\.txt`

  will match any file with a .txt extension such as example.txt.

12	**Remove File Pattern Button**
	Click to remove a file pattern.

Creating a Rule Set

To create a new rule set:

1. Click on the name of an Environment, and then click the **Rule Set** tab.

2. In the upper right-hand corner of the **Rule Set** screen, click **Create Rule Set**.

3. The **Create Rule Set** screen lets you specify which tables belong in the rule set.

4. Enter a **name** for the new Rule Set.

5. Select a **Connector** name from the drop-down menu.

6. The list of tables for that connector appears. If you have not yet created any connectors, the list is empty. Click individual table names to select them, or click **Select All** to select all the tables in the connector. See "Create/Edit Rule Set Window" for a description of the screen and other options.

7. Click **Save**.

You may then need to define the Rule Set by modifying the table settings as described in "Modifying Tables in a Rule Set" below.

For example:

- For a table in a database rule set, you may want to filter data from the table.
- For a file in a file or mainframe rule set, you must select a File Format to use.

Refreshing a Rule Set

Refreshing a rule set will result in the columns in the tables in the rule set being rescanned. As a result, the inventory associated with the rule set will also be refreshed, but any pre-existing algorithm assignments will be retained.

To refresh a rule set:

1. Click the **Refresh/Save** icon to the right of the rule set on the **Rule Set** screen.

2. The **Refresh/Save** icon will turn to an hourglass as the associated tables are rescanned.

3. After the refresh is complete, the **Refresh/Save** icon will return to the circular arrow.

Copying a Rule Set

If you copy a Rule Set, the inventory associated with that Rule Set will also be copied. Also, any filter conditions defined for that Rule Set will be copied.

To copy a rule set:

1. Click the **Copy** icon to the right of the rule set on the **Rule Set** screen.

2. The **Copy Rule Set** window appears.

3. Enter a **Name** for the new rule set.

4. Click **Save**.

5. Modify the rule set as you want, using the procedures described above.

Deleting a Rule Set

If you delete a Rule Set, the inventory associated with that Rule Set will also be deleted. Also, any filter conditions defined for that Rule Set will be deleted.

To delete a rule set, click the **Delete** icon to the right of the rule set on the **Rule Set** screen.

The Rule Set Screen

From the **Rule Set** tab, click on a rule set to display the tables or files in the rule set. The **Rule Set** screen appears.

The **Rule Set** screen contains the following information and actions:

- **Table** or **File or Pattern** — The name of the table or file/file pattern in the rule set.

- **Edit** — Edit the table or file in the rule set. See more details below.

- **Delete** — Delete the table or file from the rule set.

For rule sets with a large number of tables or files, the **Rule Set** screen will be displayed on pages that can be navigated by the controls at the bottom of the list on the page. The tables or files displayed may also be filtered using the **Search** field and button.

Editing/Modifying a Rule Set

To edit a rule set:

1. Click the **Edit** icon to the right of the rule set on the Rule Set screen.

2. Click the **Edit Rule Set** button towards the top.

3. The **Create Rule Set** screen appears. This screen lets you specify which tables belong in the rule set.

4. Modify the rule set as you want, using the preceding procedures.

Removing a Table or File

To remove a table or file from a rule set:

1. From the **Rule Set** screen, click the **name** of the desired rule set.

2. Click the red **delete** icon to the right of the table or file you want to remove.

> ✎ **INFO**
>
> If you remove a table/file from a rule set and that table/file has an inventory, that inventory will also be removed.

Modifying Tables in a Rule Set

The features in this section are disabled for file and mainframe rule sets.

You can modify tables in a rule set as follows:

Logical Key

A logical key is a unique, non-null value that identifies a row in the database.

If your table has no primary keys defined in the database, and you are using an In-Place strategy, you must specify an existing column or columns to be a logical key. This logical key does not change the target database; it only provides information to Delphix. For multiple columns, separate each column using a comma. Note: If no primary key is defined and a logical key is not defined an identify column will be created.

To enter a logical key:

1. From the **Rule Set** screen, click the **name** of the desired rule set.

2. Click the green **edit** icon to the right of the table whose filter you wish to edit.

3. On the left, select **Logical Key**.

4. Edit the text for this property. The logical key cannot be more than 1024 characters in length.

5. To remove any existing code, click **Delete**.

6. Click **Save**.

Edit Filter

Use this function to specify a filter to run on the data before loading it to the target database.

To add a filter to a database rule set table or edit a filter:

1. From the **Rule Set** screen, click the **name** of the desired rule set.

2. Click the green **edit** icon to the right of the table you want.

3. On the left, select **Edit Filter**.

4. Edit the properties of this filter by entering or changing values in the **Where** field.

Be sure to specify column name with table name prefix (for example, customer.cust_id \<1000).

1. To remove an existing filter, click **Delete**.

2. Click **Save**.

Custom SQL

Use this function to supply a customized SQL SELECT Query for the table. Typically, this query will include a **WHERE** clause to filter or subset the data.

> ⚠️ **Warning**
>
> The custom SQL must contain the primary key column (or columns if the table uses a composite primary key) and all columns that will be masked.

To add or edit SQL code:

1. From the **Rule Set** screen, click the **name** of the desired rule set.

2. Click the green **edit** icon to the right of the table you want.

3. On the left, select **Custom SQL**.

4. Enter the custom SQL code for this table.

Delphix will run the query to subset the table based on the SQL you specify.

1. To remove any existing code, click **Delete**.

2. Click **Save**.

Table Suffix

If you have tables with names that change monthly, for example, tables that are appended with the current date, you can set a table suffix for a rule set.

To set a table suffix for a rule set:

1. In the **Rule Set** screen, click the **name** of the desired rule set.

2. Click the green **edit** icon to the right of the table for which you wish to set the suffix.

3. On the left, select **Table Suffix**.

4. The **Original Table Name** will already be filled in.

5. (Optional) Enter a **Suffix date Pattern** (for example, mmyy).

6. (Optional) Enter a **Suffix Value** if you want to append a specific value.

7. (Optional) Enter a **Separator** (for example, _). This value will be inserted before the suffix value (for example, tablename_0131).

8. Click **Save**.

Add Column

Use this function to select a column or columns from a table when you don't want to load data to all the columns in a table.

To add a column to a database rule set table or edit a column:

1. From the **Rule Set** screen, click the **name** of the desired rule set.

2. Click the green **edit** icon to the right of the table you want.

3. On the left, select **Add Column**.

4. Select one or more **column names** to include in the table. To remove a column, deselect it.

5. You can also choose **Select All** or **Select None**.

6. Select **Save**.

Join Table

Use this function to specify a SQL join condition so that you can define primary key/foreign key relationships between tables.

To define or edit the join condition for a table:

1. From the **Rule Set** screen, click the **name** of the desired rule set.

2. Click the green **edit** icon to the right of the table you want.

3. On the left, select **Join Table**.

4. Edit the properties for this join condition.

5. To remove an existing join condition, click **Delete**.

6. Click **Save**.

List

Use this function to select a list to use for filtering data in a table.

To add or edit a list:

1. From the **Rule Set** screen, click the **name** of the desired rule set.

2. Click the green **edit** icon to the right of the table you want.

3. On the left, select **List**.

4. Edit the text file properties for this list.

   a. Select a **column**.

   b. Enter or browse for a **filename**.

   c. Files that have already been specified appear next to **Existing File**.

5. To remove an existing list file, click **Delete**.

6. Click **Save**.

Creating a Ruleset For File Formats

Once you create a ruleset with a file or set of files, you will need to assign those files to their appropriate file format.

This is accomplished by editing the ruleset. Click on the edit button for the file the Edit File window will appear with the file name. From the format drop-down select the proper format for the file.

- If the file is a Mainframe data sets file with a copybook you will see a checkbox to signify if the file is variable length.

- For all other file types, select the end-of-record to let Delphix know whether the file is in windows/dos format (CR+LF) or Linux format (LF).

- If the file is a delimited file you will have a space to put in the delimiter.

- If there are multiple files in the ruleset you will have to edit each one individually and assign it to the appropriate file format.

Control Character Support for Delimited Files

The user can specify control character as a delimiter/end of record from UI/API.

> ✏️ **NOTE - Control Character**
>
> The control character value from UI/API should be in **$[hex value of the control character]** format , like $[01] for ^A.
> The control character value support UTF-8 character set.
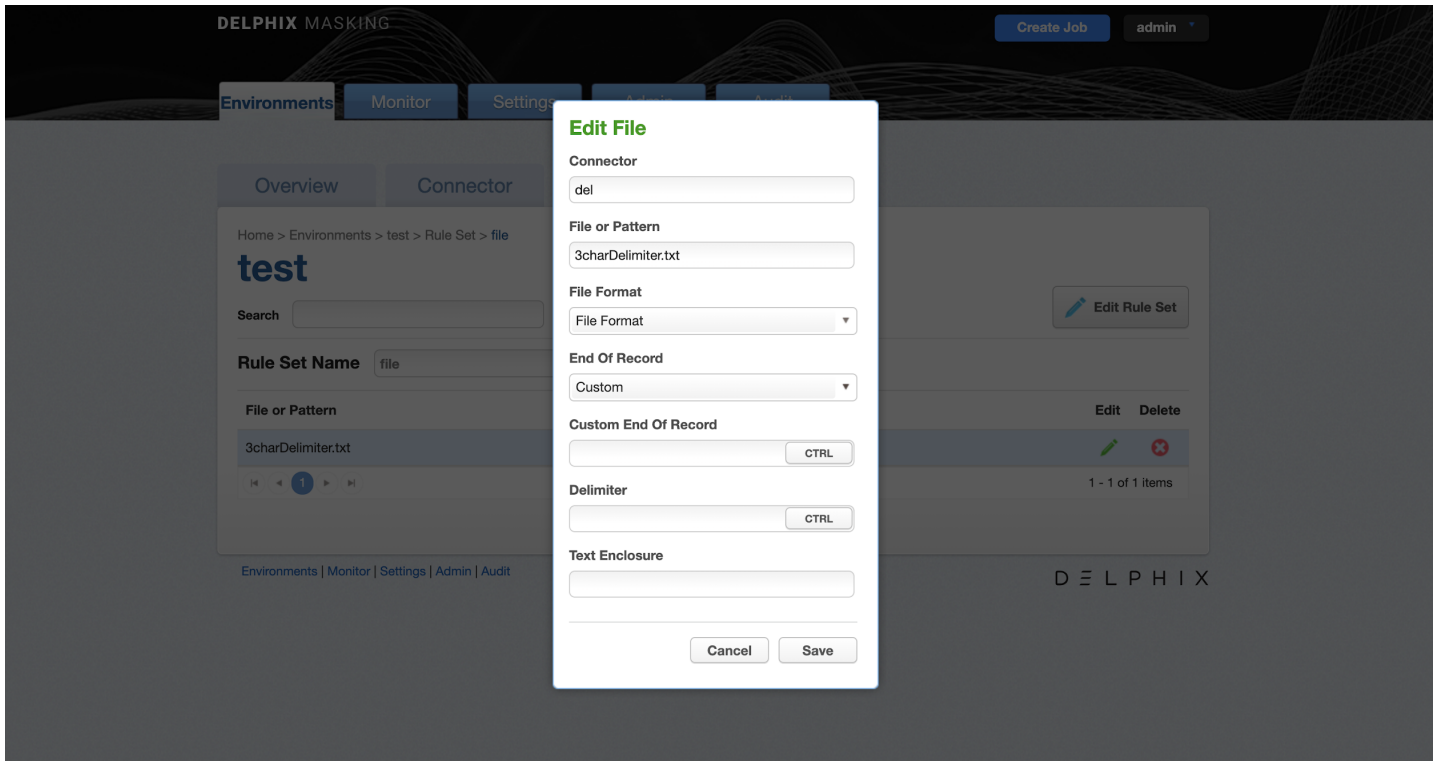
Control character as a delimiter

1. In order to use control character as a delimiter, the user needs to click on **CTRL** button inside delimiter input text.

2. Clicking on **CTRL** button will open a virtual keyboard where users can select the required control character. Also if the user wants to enter the control character manually then they can use the given format **$[hex value of the control character]** , like $[01] for ^A.
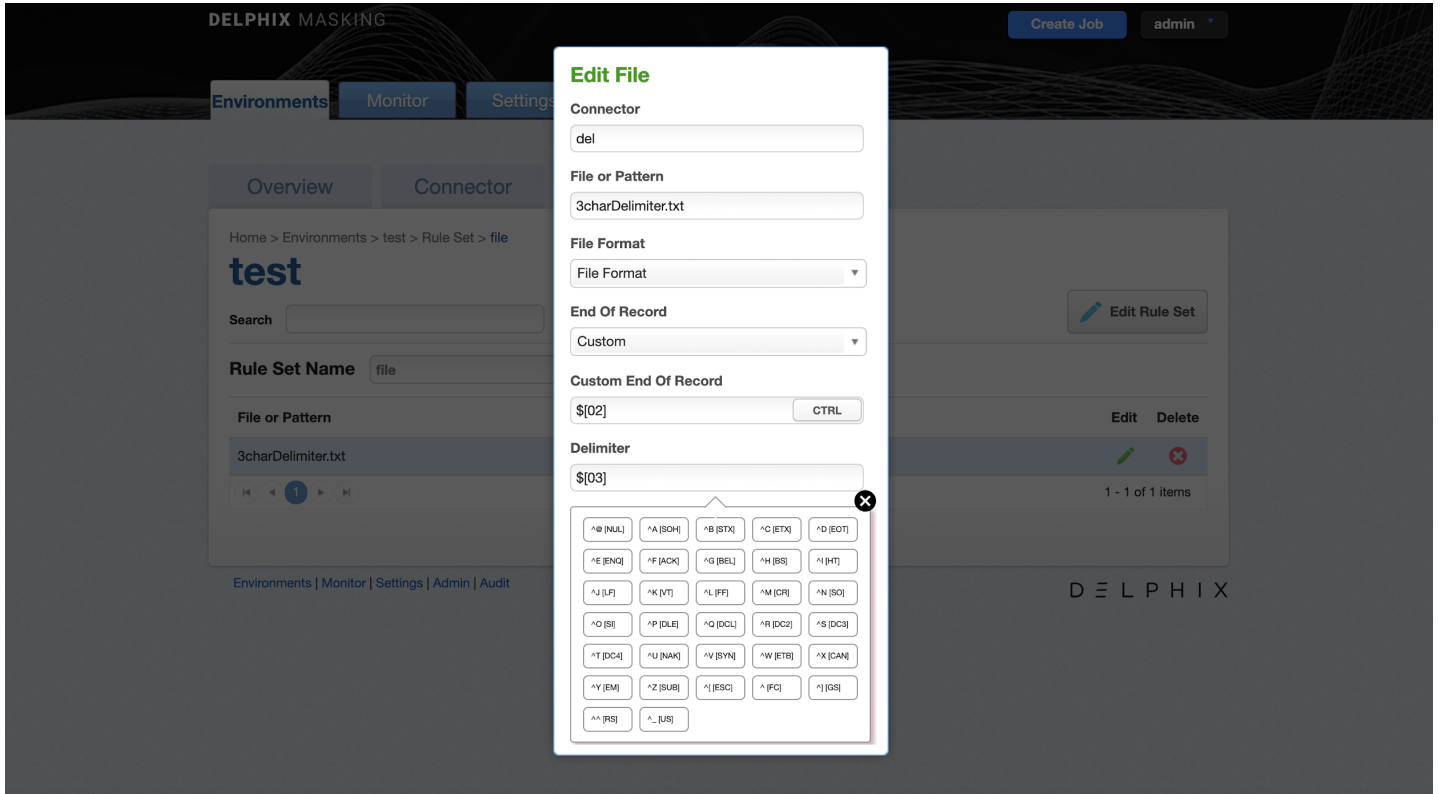
Control character as an end of record

1. In order to use control character as an end of record, the user needs to click on **CTRL** button inside custom end of record input text.

2. Clicking on **CTRL** button will open a virtual keyboard where users can select the required control character. Also if the user wants to enter the control character manually then they can use the given format **$[hex value of the control character]** , like $[01] for ^A.
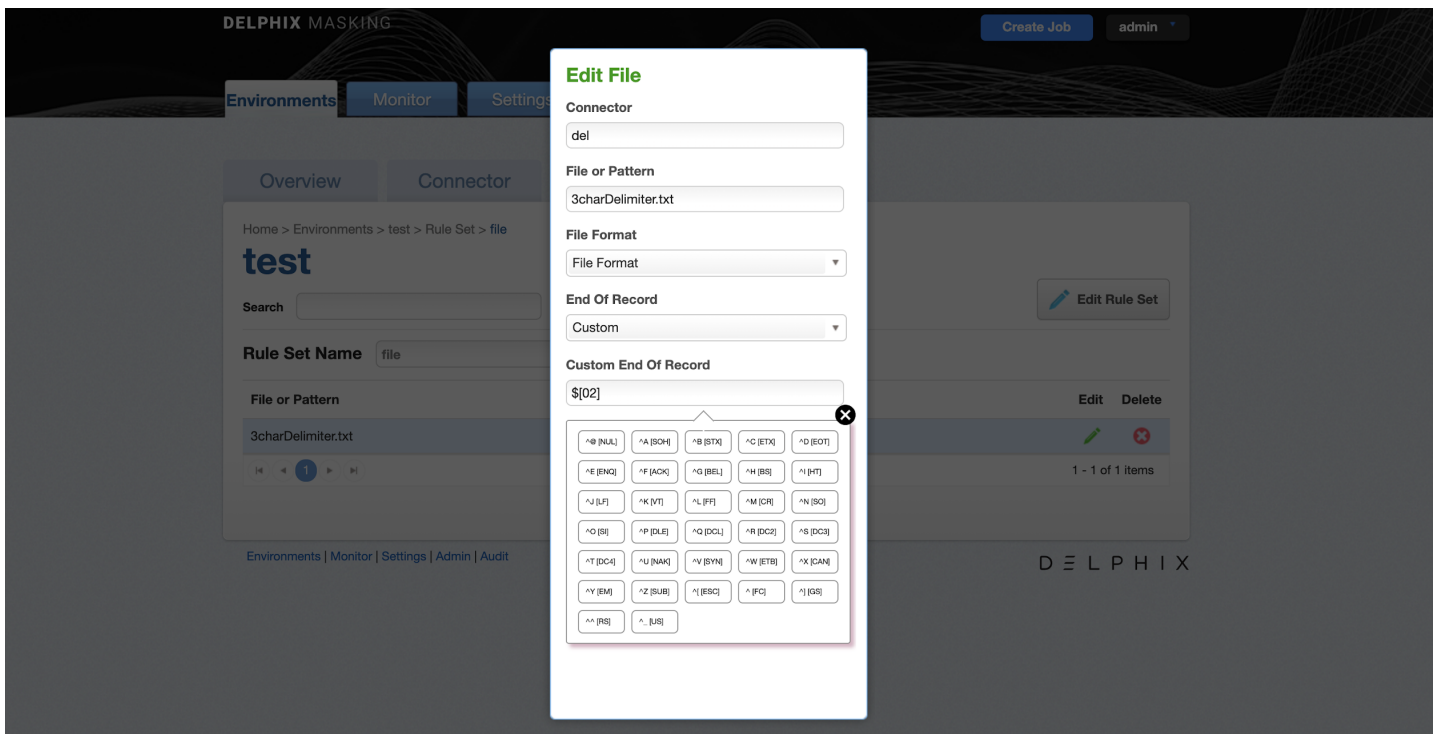
Control character as a value

1. Control characters are supported as values in a delimited file. No special configuration is necessary. Simply configure the deliited file format as usual.

2. The user doesn't need to configure anything extra if the control character is only part of the value and not being used as a delimiter or end of record. However, the user needs to define delimiter/end of record as per the requirement.

Define Enclosure Escaping Strategy for Delimited Files

The user can configure the enclosure escape character from the UI/API to escape the enclosure. To configure the enclosure escape character from the UI, user needs to select the "Enclosure Escaping Strategy" dropdown value as per below options on the edit ruleset popup window,

1. Double Enclosure

Double enclosure option will set the escape character value same as enclosure value. For example, if the enclosure escape character is " then escape character value will be " as well.
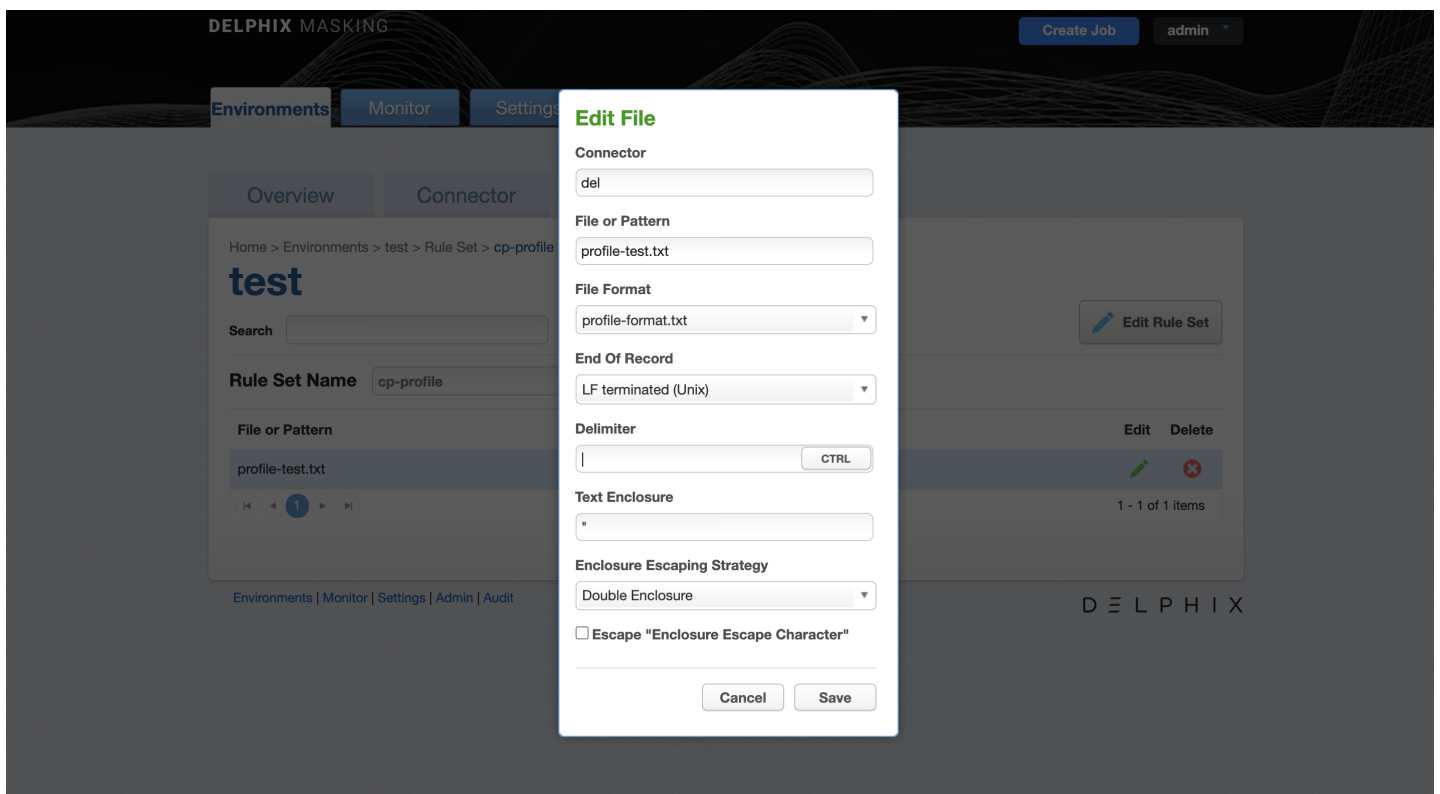
2. Custom

By selecting custom option user can specify any single character as an enclosure escape character except the "escape sequences" and "control characters".
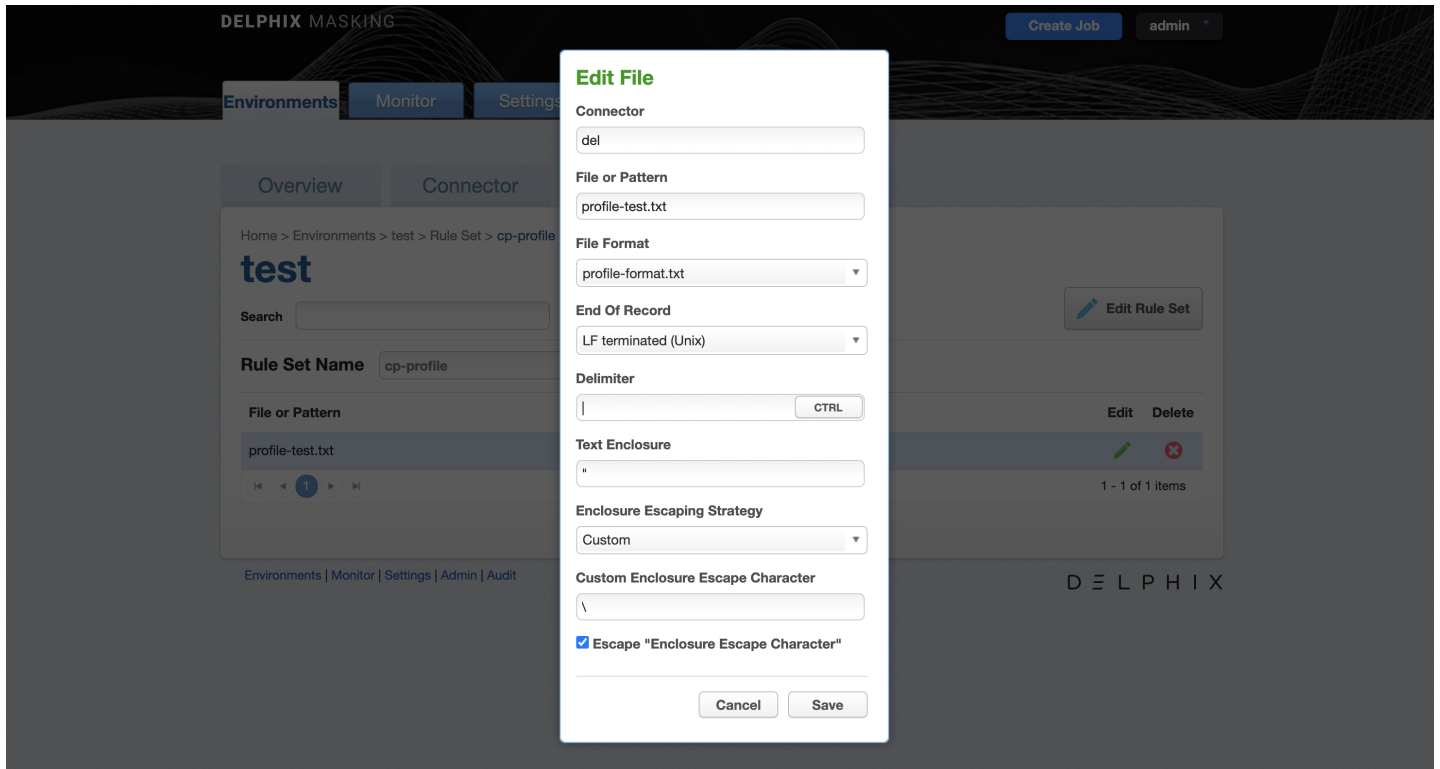
> ✎ **NOTE - Default Enclosure Escape Character**
>
> The default value for "Enclosure Escaping Strategy" is "Double Enclosure".

Escape "Enclosure Escape Character"

Selecting this checkbox indicates whether the enclosure escape character also escapes itself. For example, if the enclosure escape character is " then the sequence "" would be treated as a single " character, rather than an escape.

Configure enclosure escape character for the large ruleset

To configure the enclosure escape character for the large ruleset user can use this API Script.

Managing File Formats

File formats

Unlike database files for the most part do not have built-in metadata to describe the format of the fields in the file. You must provide this to Delphix so it can update the file appropriately. This is done through the settings tab where you will see a menu item on the left for File Format. Select File Format and you will see an option to import a file format. This will depend on the type of file and how you want to let Delphix know the format of the file.
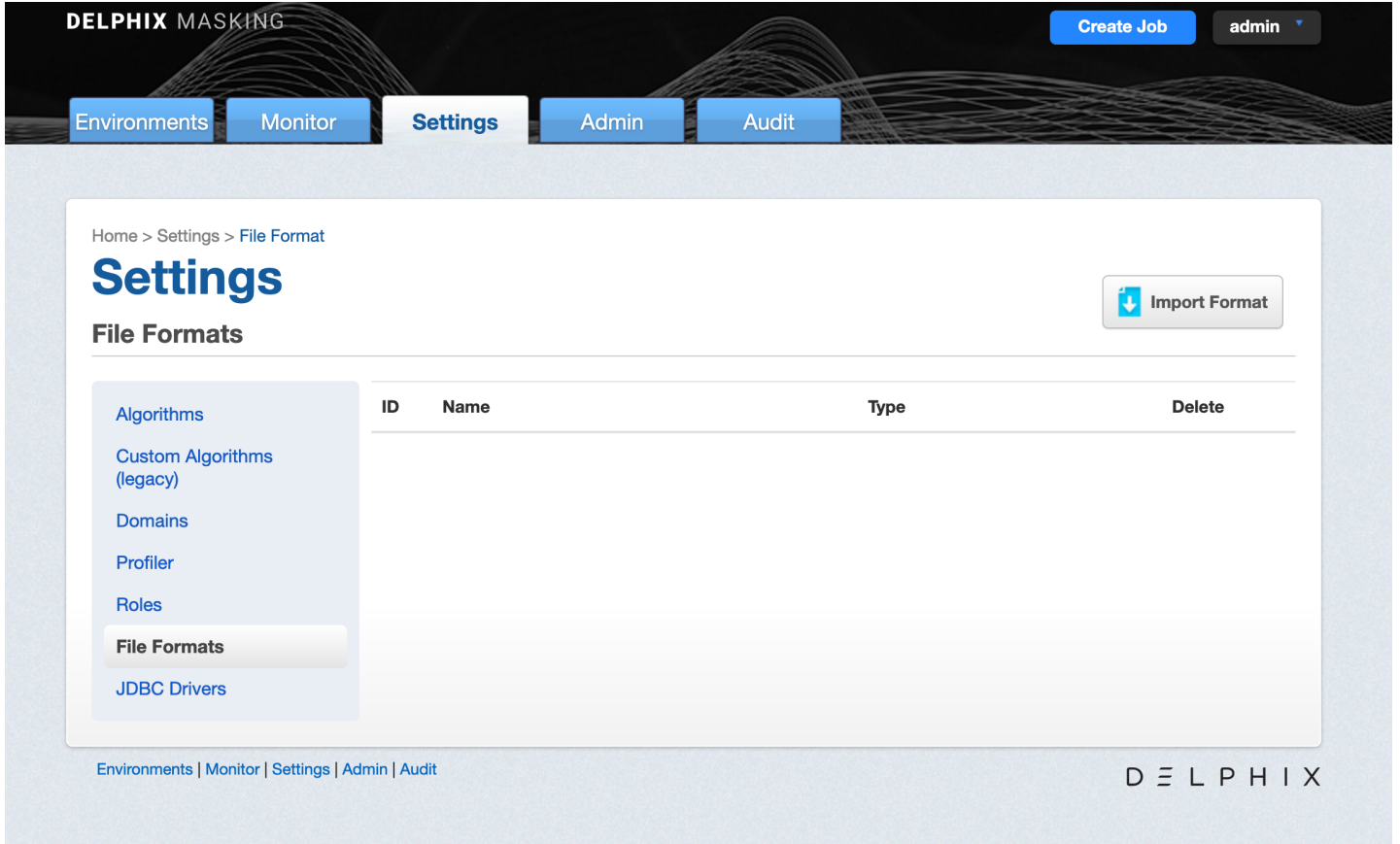
Mainframe data sets and XML Files

For Mainframe data sets, you can specify the file format via the Import Format button which will import the copybook directly into Delphix. You can input this file from a Filesystem Mount Point, SFTP server, FTP server, or via upload. Please select Copybook as the Import Format Type.

For XML files you can also import the file format with the input format option which will import the file directly into Delphix. You can use the file you want to mask as the format. You can input this file from a Filesystem Mount Point, SFTP server, FTP server, or via upload. Please select XML as the Import Format Type.

Delimited and Fixed files

For Delimited and Fixed files you can import a text file that describes the structure of the file to Delphix.

To input the file format for delimited files, create a text document with the column names each on its own line. For example:

- Name

- Address

- City

- State

To input the file format for fixed files, create a text document with the column names and the length of each column on its own line. For example:

- Name,25

- Address,40

- City,20

- State,2

Then input this file as the file format. The name of the text file will be the name of the file format.

> ✎ **NOTE - Column length Mismatch between Fixed File and File Format**
>
> For Fixed Files, caution should be taken to ensure that the column length is in accordance with the File Format definition. Failure to do so will result in masking a column with the incorrect offset, which would have the unintended consequence of not masking what was intended.

> ✎ **NOTE - Behavior when the number of fields in a delimited file's format and contents are mismatched**
>
> The behavior is this case is as follows:
>
> 1. If the total number of fields in the Delimited File is less than the total number of fields in the File Format, then after masking, delimiter will be added to match the total fields with File Format. See the below example,
>   **Format:** One, Two, Three
>   **Delimited File Data:** Test Data1, Test Data2
>   **Result after masking:** Test Data1, Test Data2, (One extra delimiter will be added to match with the File Format column length).
>
> 2. If the total number of fields in the Delimited File is greater than the total number of fields in the File Format, then after masking the extra fields in the Delimited File will be lost. See the below example,
>   **Format:** One, Two, Three
>   **Delimited File Data:** Test Data1, Test Data2, Test Data3, Test Data4
>   **Result after masking:** Test Data1, Test Data2, Test Data3

> ✎ **NOTE - Multi-byte Characters**
>
> For Fixed Files, column length is determined by the number of characters rather than the number of bytes.

To Import a New File Format

1. Click **Import Format** at the upper right. The Import File Format window appears.

2. Select an **Import File Type**.

For a Format Type of Copybook or XML
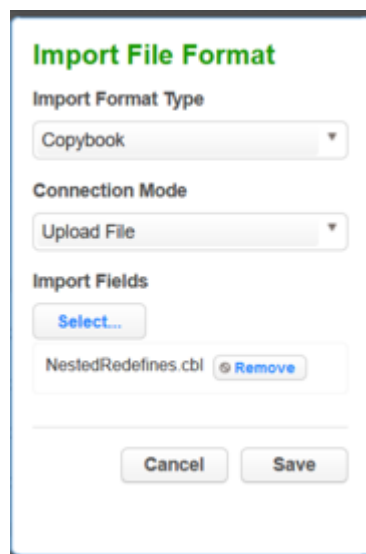
1. Select a **Connection Mode**.

2. Fill out the required fields of the selected **Connection Mode**. For Filesystem Mount Point connection mode, refer to the Managing Remote Mounts page to fill out the required fields.

3. Click **Browse**.

4. Click the **Select** button to the right of the desired import file format.

5. Enter a **Logical Name**.

6. Click **Submit**.

For a Format Type of Delimited File, or Fixed Width File

1. Click **Select**.

2. Browse for the file from which to import fields.

3. Click **Save**.

Note: - The file must have NO header. - Make sure there are no spaces or returns at the end of the last line in the file. - To be masked, the field names must be in the same order as they are in the file.

Removing a Selected File

If you accidentally selected an incorrect file, simply click the Remove button to the right of the file and repeat the selection steps above.

Samples

The following is sample file content for Delimited file formats. With these formats, just the field name is provided. Notice there is no header and only a list of values.

```
First_Name
Last_Name
DOB
SSN
Address
City
State
Zip_Code
```

The following is sample file content for Fixed Width format. In this format, the field name is followed by the length of the field, separated by a comma. Notice there is no header and only a list of values.

```
First_Name,20
Last_Name,30
DOB,10
SSN,11
Address,30
City,20
State,2
Zip_Code,10
```

To Delete a File Format

1. Click the **Delete** icon to the right of the File Format name.

2. File inventory is based on file format. Therefore, if you make a change to a file inventory, that change applies to *all* files that use that format.

3. You can only add or delete a file format; you cannot edit one.

Assigning a File Format to a files

Once you create a rule set with a file or set of files, you will need to assign those files to their appropriate file format. This is accomplished by editing the rule set. When you click on the edit button for the file a pop-up screen called edit file will appear with the file name. There will be a drop-down for the format so you can select the proper format for the file. If the file is a Mainframe data sets file with a copybook you will see a checkbox to signify if the file is variable length. For all other file types, select the end-of-record to let Delphix know whether the file is in windows/dos format (CR+LF) or Linux format (LF). If the file is a delimited file you will have a space to put in the delimiter. If there are multiple files in the ruleset you will have to edit each one individually and assign it to the appropriate file format.

Managing Inventories

An inventory describes all of the data present in a particular ruleset and defines the methods which will be used to secure it. Inventories typically include the table or file name, column/field name, the data classification, and the chosen algorithm.

The Inventory Screen

From anywhere within an environment, click the **Inventory** tab to see the Inventory Screen. This displays the inventory for the environment's rule sets.

Inventory Settings

To specify your inventory settings:

1. On the left-hand side of the screen, select a **Rule Set** from the drop-down menu.

2. Below this, **Contents** lists all the tables or files defined for the ruleset.

3. Select a **table** or **file** for which you want to create or edit the inventory of sensitive data. The **Columns** or **Fields** for that specific table or file appear.

4. If a column is a primary key (PK), Foreign Key (FK), or index (IDX), an icon indicating this will appear to the Right of the column name. If there is a note for the column, a Note icon will appear. To read the note, click the icon.

5. If an algorithm associated with a column is a custom algorithm (formerly known as Mapplet) then **(*custom)** in red text will appear after the algorithm name.

6. If you selected a table, metadata for the column appears: **Data Type** and **Length** (in parentheses). This information is read-only.

7. Choose how you would like to view the inventory:

   - **All Fields** — Displays all columns in the table or all fields in the file (allowing you to mark new columns or fields to be masked).

   - **Masked Fields** — Filters the list to just those columns or fields that are already marked for masking.

   - **Auto** — The default value. The profiling job can determine or update the algorithm assigned to a column and whether to mask the column.

   - **User** — The user's choice overrides the profiling job. The user manually updates the algorithm assignment, mask/unmask option of the column. The Profiler will ignore the column, so it will not be updated as part of the Profiling job.

Assigning Algorithms

To set criteria for sensitive columns or fields:

1. Click the green edit icon to the right of a column or field name.

2. From the Domain drop-down menu, select the appropriate sensitive data element type.

3. The Delphix Masking Engine defaults to a **Masking Algorithm** as specified in the Settings screen. If necessary, you can override the default algorithm.

   - To select a different masking algorithm, choose one from the Algorithm dropdown.

   - In the algorithm pulldown, any custom algorithms will appear with **(*custom)** after their name to make them easier to identify. For detailed descriptions of these algorithms, please see Out Of The Box Algorithm Frameworks.

4. Select an ID Method:

   - **Auto** — The default value. The profiling job can determine or update whether to mask a column.

   - **User** — The user decides whether to mask/unmask a column. The user's choice overrides the profiling job. (The user masking is done after the profiling job is finished.)

5. You can add/remove notes in the **Notes** text field.

6. When you are finished, click **Save**. You must click Save for any edits to take effect.

> ✏️ **Note**
>
> If you select a DATESHIFT algorithm and you are not masking a datetime or timestamp column, you must specify a **Date Format**. (This field only appears if you select a DATESHIFT algorithm from the Masking Algorithm dropdown.) For a list of acceptable formats, click the **Help** link for Date Format. The default format is yyyy-MM-dd.

Managing a File Inventory

Defining fields

To create new fields:

1. From an Environment's Inventory tab, click **Define fields** to the far right. The Edit Fields window appears.

2. Edit the fields as described in **Setting Field Criteria for a File**.

3. When you are finished, click **New** to create a new field, or click **Save** to update an existing field.

Adding Record Types for files

To add a new Record Format:

1. In the upper right-hand corner of an environment's **Inventory** tab, click **Record Types**. The Record Type window appears.

2. Click **+Add a Record Type** towards the bottom of the window. The Add Record Type window appears.

3. Enter values for the following fields:

   - **Record Type Name** — A free-form name for this record format.

   - **Header/Body/Trailer** — If the file has header or trailer records, you will need to create file formats for them. Select the appropriate type. Delphix allows for masking of multiple headers, multiple trailers, and multiple types of body records.

   - **Record Type ID** — (optional) For body records, specify the value of the record type code or another identifier that allows Delphix to identify records that qualify as this record type.

- **Position #** — (optional) Specify the field number (for delimited files) or the character position number (for fixed files) of the beginning of the Record Type Identifier within the data record.

- **Length #** — (optional) For fixed files, specify the length of the Record Type Identifier within the data record.

4. Click **Save** when you are finished.

Managing a Mainframe Inventory

Redefine Conditions

For Mainframe data sets, the inventory also allows for the entry of Redefine Conditions, which are used to handle any occurrences of COBOL's REDEFINES construct that might appear in the Copybook. In COBOL, the REDEFINES keyword allows an area of a record to be interpreted in multiple different ways. In the example below, for instance, each record can hold either the details of a person (PERSON-DET) or the details of a company (COMP-DET).

```
01   CS-CUSTOMER-RECORD.
   05   CUST-TYPE                      PIC X(1).
   05   PERSON-DET.
      10   PERSON-FIRSTNAME            PIC X(20).
      10   PERSON-LASTNAME             PIC X(40).
      10   PERSON-ADDRESS1             PIC X(50).
      10   PERSON-CITY                 PIC X(20).
      10   PERSON-STATE                PIC X(5).
      10   PERSON-ZIP                  PIC X(10).
      10   PERSON-SSN                  PIC S9(9) COMP-3.
   05   COMP-DET                       REDEFINES PERSON-DET.
      10   COMP-ENTITYNM               PIC X(53).
      10   COMP-ADDRESS1               PIC X(50).
      10   COMP-CITY                   PIC X(20).
      10   COMP-STATE                  PIC X(5).
      10   COMP-ZIP                    PIC X(10).
      10   COMP-PHONE                  PIC X(12).
```

Depending on which group is present, different masking algorithms may need to be applied. Below is the inventory corresponding to this copybook, which allows algorithms to be selected separately for each group.
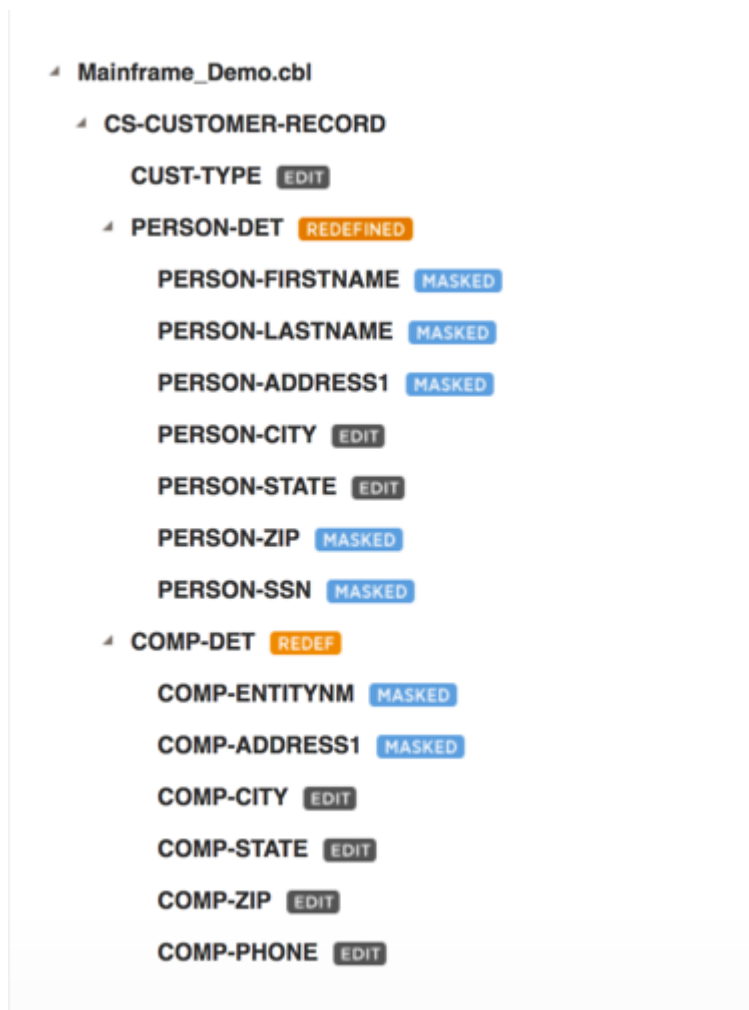
In order to do any masking however, the Masking Engine must be able to determine, for each record, which fields should be read, so that the correct algorithms can be applied. In order to do this, the masking engine uses Redefine Conditions, which are specified in the inventory. Redefine Conditions are boolean expressions which can reference any fields in the record when they are evaluated.

In the example copybook above, the field CUST-TYPE is used to indicate which group is present. If CUST-TYPE holds a 'P', a PERSON-DET group is present, and if it holds a 'C', COMP-DET is present. This can be expressed in the inventory by specifying a Redefine Condition with the value [CUST-TYPE]='P' . This expression indicates that, for each record read from the source file during the masking job, the value of the field CUST-TYPE should be read and compared against the string 'P'. If it is equal, the Masking Engine will read from the record the fields subordinate to PERSON-DET, and will apply any masking algorithms specified on those fields. Similarly, a Redefine Condition with the value [CUST-TYPE]='C' should be applied to the COMP-DET field. Exactly one of the conditions should evaluate to 'true' for each group of redefined fields. For example, a copybook might have fields A, B REDEFINES A, and C REDEFINES A. Of the Redefine Conditions attached to A, B, and C, one and only one should evaluate to true for each record.

Entering a Redefine Condition

1. Click on the orange **REDEFINED** or **REDEF** button next to the redefined or redefining field

2. Enter a condition in the dialog box which appears. This is the expression, which, when it evaluates to true, causes the subordinate fields to be read and, if they have algorithms assigned, masked.
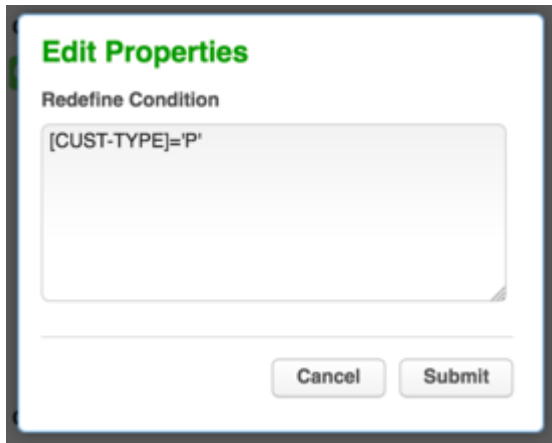
3. Click **Submit**.

Format of Redefine Conditions

Redefine Conditions allow fields to be compared against either number or string literals. Square brackets enclosing a field name indicate a variable, which takes on the value of the named field:

```
[Field1] = 'An example String'
```

String literals can be enclosed in either single or double quotes. For fields that are numeric (e.g. PIC S99V9), the operators <, <=, >, and >= can be used in addition to the =operator, e.g.

```
[Field2] <= -10.5
```

Also, conditions can be joined using AND, OR, and NOT to form more complex conditions:

```
([Field3] > 2.5 AND [Field3] < 10) OR NOT [FIELD4] = 'Z'
```

Importing and Exporting an Inventory

To export an inventory:

1. Click the **Export** icon at the upper right. The Export Inventory pop-up appears with the name of the currently selected Rule Set as the Inventory Name and a corresponding .csv **File Name**.
2. Click **Save**.

A status pop-up appears. When the export operation is complete, you can click on the **Download file** name to access the inventory file

To import an inventory:

1. In the upper right-hand corner, click the **Import** icon. The Import Inventory pop-up appears.
2. Click **Select** to browse for the name of a comma-separated (.csv) file.
3. Click **Save**.

The inventory you imported appears in the Rule Set list for this environment.

> **ⓘ Info**
>
> You can import only one ruleset at a time.

> **ⓘ Info**
>
> The format of an imported.csv file must exactly match the format of the exported inventory. If you plan to import an inventory, before importing the inventory, you should export it and then update the exported file as needed before you import it.

Introduction

This feature offers standard functionalities for masking JSON files. Users will now be able to configure and run Continuous Compliance jobs specific to JSON files, assigning algorithms to any field of a JSON file using their respective JSON paths. This feature overcomes the shortfalls of the existing algorithm-based workaround by providing users with a simplified way to assign Continuous Compliance algorithms. This feature also supports masking JSON files of large sizes.

These features are not yet supported:

- Profiling Job for JSON File Rulesets

- Tokenization and Re-Identification for JSON File Rulesets

- Multi-Column Algorithms for JSON File Formats

API Changes

API	Change Description
POST /file-formats	Added support to upload a Json file to create JSON File Format.
PUT /file-formats/{fileFormatId}	Added validations to stop creating headers and footers for JSON File Formats.
POST /file-connectors	Added support to create a new file connector of type `File - JSON`.
POST /file-field-metadata	Added support to create a new JSON File field, specifying its JSON path identifier and assigning algorithms to it.
PUT /file-field-metadata	Added support to update JSON File field to assign or unassign algorithms to it.

GUI Changes

In the Continuous Compliance interface, navigate to **Settings > File Format**. Import the JSON file to create JSON File Formats.

In the Create Connection screen, choose **File - JSON** from the Type dropdown and configure the appropriate details.

The **Inventory** tab for JSON File Formats is used to configure algorithms to JSON Paths.

Navigate to **Monitor > Processing** to access the Job Process Monitoring page. This page shows data in byte format for JSON file masking.

Constructing a JSON File Path

A JsonPath expression begins with the dollar sign ($) character, which refers to the root element. The dollar sign is followed by a sequence of child elements, which are separated by the square brackets ([''']) containing the name of each JSON field. If the field is inside an array, a star character is used to represent all elements of the array ([*]).

Media

Introduction

This feature offers standard functionalities for masking JSON files. Users will now be able to configure and run Continuous Compliance jobs specific to JSON files, assigning algorithms to any field of a JSON file using their respective JSON paths. This feature overcomes the shortfalls of the existing algorithm-based workaround by providing users with a simplified way to assign Continuous Compliance algorithms. This feature also supports masking JSON files of large sizes.

These features are not yet supported:

- Profiling Job for JSON File Rulesets

- Tokenization and Re-Identification for JSON File Rulesets

- Multi-Column Algorithms for JSON File Formats

API Changes

API	Change Description
POST /file-formats	Added support to upload a Json file to create JSON File Format.
PUT /file-formats/{fileFormatId}	Added validations to stop creating headers and footers for JSON File Formats.
POST /file-connectors	Added support to create a new file connector of type `File - JSON`.
POST /file-field-metadata	Added support to create a new JSON File field, specifying its JSON path identifier and assigning algorithms to it.
PUT /file-field-metadata	Added support to update JSON File field to assign or unassign algorithms to it.

GUI Changes

In the Continuous Compliance interface, navigate to **Settings > File Format**. Import the JSON file to create JSON File Formats.

[ImportJSON.png]

In the Create Connection screen, choose **File - JSON** from the Type dropdown and configure the appropriate details.

[CreateConnection.png]

The **Inventory** tab for JSON File Formats is used to configure algorithms to JSON Paths.

[InventoryTab.png]

Navigate to **Monitor > Processing** to access the Job Process Monitoring page. This page shows data in byte format for JSON file masking.

[MonitoringPage.png]

The example below shows the Kettle Step configuration in the **Character Streaming File Input Step**.

[KettleStep.png]

Constructing a JSON File Path

A JsonPath expression begins with the dollar sign ($) character, which refers to the root element. The dollar sign is followed by a sequence of child elements, which are separated by the square brackets ([''']) containing the name of each JSON field. If the field is inside an array, a star character is used to represent all elements of the array ([*]).

[JSONbracket.png]

Identifying Sensitive Data

Discovering Your Sensitive Data

After connecting data to the masking service, the next step is to discover which of the data should be secured. This sensitive data discovery is done using two different methods, column-level profiling, and data level profiling.

Column Level Profiling

Column level profiling uses regular expressions (regex) to scan the metadata (column names) of the selected data sources. There are several dozen pre-configured profile Expressions (like the one below) designed to identify common sensitive data types (SSN, Name, Addresses, etc). You also have the ability to write your own profile Expressions.

First Name Expression <([A-Z][A-Z0-9])\b[^>]>(.*?)</\1>

Data Level Profiling

Data level profiling also uses regex, but to scan the actual data instead of the metadata. Similar to column level profiling, there are several dozen pre-configured Expressions (like the one below) and you can add your own.

Social Security Number Expression <([A-Z][A-Z0-9])\b[^>]>(.*?)</\1>

For both column and data level profiling, when a data item is identified as sensitive, Delphix recommends/assigns particular masking algorithms to be used when securing the data. The platform comes with several dozen pre-configured algorithms which are recommended when the profiler finds certain sensitive data.

Out of the Box Profiling Settings

The Delphix Platform comes out of the box with over 50 profile Expressions to help you discover over 30 types (account numbers, addresses, etc.) of sensitive data.

Account Numbers

An account number is the primary identifier for ownership of an account, whether a vendor account, a checking or brokerage account, or a loan account. An account number is used whether or not the identifier uses letters or numbers. Below are the profile Expressions Delphix uses to identify account numbers:

Expression Name	Domain	Expression Level	Expression
Account Number	ACCOUNT_NO	Column	`(?>(acc(oun|n)?t)_?(num(ber)?|nbrjno)?)(?!\w\*(ID|type))`

Physical Addresses

Below are the profile Expressions Delphix uses to identify physical addresses:

Expression Name	Domain	Expression Level	Expression
Address	ADDRESS	Column	`^(?:(?!postalcode|city|state|country|email|(l|ln|lin|line)?_?2{1}|ID).)*addre?s?s?_?(?:(?!city|state|country|email|(l|ln|lin|line)?_?2{1}|ID).)*$`
Street Address	ADDRESS	Column	`(?>(str(eet)?_?addre?s?s?|street))(?!\w*(ID|type))`
Data - Address	ADDRESS	Data	`(.*[\s]+b(ou)?|(e)?v(ar)?d[\d]*.*)|(.*[\s]+st[.]?(reet)?[\s]*.*)|(.*[\s]+ave[.]?(nue)?[\s]*.*)|(.*[\s]+r(oa)?d[\s]*.*)|(.*[\s]+|(a)?n(e)?[\s]*.*)|(.[\s]+cir(cle)?[\s]*.*1`
Address Line2 - before	ADDRESS_LINE2	Column	`^(?:(?!email|ID).)*(l|ln|lin|line)?2{1}_?addre?s?s?(?:(?!email|ID).)*$`
Address Line2	ADDRESS_LINE2	Column	`^(?:(?!email|ID).)*addre?s?s?_?(l|ln|lin|line)?_?`

Expression Name	Domain	Expression Level	Expression
- after			`2{1}(?:(?!email|ID).)*$`
Data - Address Line 2	ADDRESS_LINE2	Data	`(.*[\s]*ap(ar)?t(ment)?[\s]+.*)|(.*[\s]*s(ui)?te[\s]+.*)|(c(are)?[\s]*[\\\\]?[/]?o(f)?[\s]+.*)`

Beneficiary ID

Below are the profile Expressions Delphix uses to identify beneficiary IDs:

Expression Name	Domain	Expression Level	Expression
Beneficiary Number	BENEFICIARY_NO	Column	`(?>(bene(ficiary)?)_?(num(ber)?|nbr|no))(?!\w*ID)1`
Beneficiary ID	BENEFICIARY_NO	Column	`(?>(bene(ficiary)?)_?id)`

Biometrics

Below are the profile Expressions Delphix uses to biometric data:

Expression Name	Domain	Expression Level	Expression
Biometric	BIOMETRIC	Column	biometric

Certificate ID

Below are the profile Expressions Delphix uses to identify certificate IDs:

Expression Name	Domain	Expression Level	Expression
Certificate Number	CERTIFICATE_NO	Column	`(?>cert(ificate)?_?(num(ber)?|nbr|no|id))`
Certificate ID	CERTIFICATE_NO	Column	`(?>cert(ificate)?_?id)`

City

Below are the profile Expressions Delphix uses to identify cities:

Expression Name	Domain	Expression Level	Expression
City	CITY	Column	`ci?ty(?!\w*ID)`

Country

Below are the profile Expressions Delphix uses to identify countries:

Expression Name	Domain	Expression Level	Expression
Country	COUNTRY	Column	`c(ou)?nty(?!\w*ID)`

Credit Card

Below are the profile Expressions Delphix uses to identify credit cards:

Expression Name	Domain	Expression Level	Expression
Card Number	CREDIT CARD	Column	`(?>ca?rd_?(num(ber)?|nbr|no)?)(?!\w*ID)`
Credit Card Number	CREDIT CARD	Column	`(?>cre?di?t_?(ca?rd)?_?(num(ber)?|nbr|no)?)(?!\w*ID)`
Data - Credit Card	CREDIT CARD	Data	`^(?:3[47][0-9]{13}|4[0-9]{12}(?:[0-9]{3})?(?:[0-9]{3})?|(?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]|2[3-6][0-9]{2}|27[01][0-9]|2720)[0-9]{12}|6(?:(011|5[0-9][0-9])[0-9]{2}|4[4-9][0-9]{3}|2212[6-9]|221[3-9][0-9]|22[2-8][0-9]{2}|229[0-1][0-9]|2292[0-5])[0-9]{10}?(?:[0-9]{3})?|3(?:0[0-5,9]|6[0-9])[0-9]{11}|3[89][0-9]{14}?(?:[0-9]{1,3})?)$`

Customer Number

Below are the profile Expressions Delphix uses to identify customer IDs:

Expression Name	Domain	Expression Level	Expression
Customer Number	CUSTOMER_NUM	Column	`(?>(cu?st(omer|mr)?)_?(num(ber)?|nbr|no)?)(?!\w*ID)`

Date of Birth

Below are the profile Expressions Delphix uses to identify dates of birth:

Expression Name	Domain	Expression Level	Expression
Birth Date	DOB	Column	`(?>(bi?rth)_?(date?|day|dt))(?!\w*ID)`
Birth Date1	DOB	Column	`(?>dob|dtofb|(day|date?|dt)_?(of)?_?(bi?rth))(?!\w*ID)`
Birth Date2	DOB	Column	`(?>b_?(date?|day))(?!\w*ID)`
Admission Date	DOB	Column	`(?>(adm(it|ission)?)_?(date?|day|dt))(?!\w*ID)`
Treatment Date	DOB	Column	`(?>(tr(ea)?t(ment)?)_?(date?|day|dt))(?!\w*ID)`
Discharge Date	DOB	Column	`(?>(ds|disc(h|harge)?)_?(date?|day|dt))(?!\w*ID)`

Driver License Number

Below are the profile Expressions Delphix uses to identify driver license numbers:

Expression Name	Domain	Expression Level	Expression
Drivers License Number	DRIVING_LC	Column	`(?>(dri?v(e?rs?e?)?)_?(license|li?c)?_?(num(ber)?|nbr|no)?)(?!\w*ID)`
Drivers License Number1	DRIVING_LC	Column	`(^license$|(license|li?c)_?(num(ber)?|nbr|no))(?!\w*ID)`

Email

Below are the profile Expressions Delphix uses to identify emails:

Expression Name	Domain	Expression Level	Expression
Email	EMAIL	Column	`^(?:(?!invalid).)*email(?!\w*ID)`
Data - Email	EMAIL	Column	`\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,6}\b`

First Name

Below are the profile Expressions Delphix uses to identify first names:

Expression Name	Domain	Expression Level	Expression
First Name	FIRST_NAME	Column	`(?>(fi?rst)_?(na?me?)|f_?name)(?!\w*ID)`
Middle Name	FIRST_NAME	Column	`(?>(mid(dle)?)_?(na?me?)|m_?name)(?!\w*ID)`

IP Address

Below are the profile Expressions Delphix uses to IP addresses:

Expression Name	Domain	Expression Level	Expression
IP Address	IP ADDRESS	Column	`(?>(ip_?addre?s?s?))(?!\w*(ID|type))`
Data - IP Address	IP ADDRESS	Data	`\b(?:(?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\.){3}(?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\b`

Last Name

Below are the profile Expressions Delphix uses to identify last names:

Expression Name	Domain	Expression Level	Expression
Last Name	LAST_NAME	Column	`^(?:(?!portal|ID).)*((la?st)_?(na?me?)|l_?name)(?:(?!portalname|ID).)*$`

Plate Number

Below are the profile Expressions Delphix uses to identify plate numbers:

Expression Name	Domain	Expression Level	Expression
License Plate	PLATE_NO	Column	`^(?:(?!template|ID|type).)*(license|li?c)?_?plate_?(num(ber)?|nbr|no)?(?:(?!template|ID|type).)*$`

PO Box Numbers

Below are the profile Expressions Delphix uses to identify PO box numbers:

Expression Name	Domain	Expression Level	Expression
PO Box	PO_BOX	Column	`po_?box`
Data - PO Box	PO_BOX	Data	`po box|p\.o\`

Precinct

Below are the profile Expressions Delphix uses to identify precincts:

Expression Name	Domain	Expression Level	Expression
Precinct	PRECINCT	Column	`(>?precinct|prcnct)(?!\w*ID)`

Record Number

Below are the profile Expressions Delphix uses to identify record numbers:

Expression Name	Domain	Expression Level	Expression
Record Number	RECORD_NO	Column	`(?>rec(ord)?_?(num(ber)?|nbr|no))(?!\w*(ID|type))`

School Name

Below are the profile Expressions Delphix uses to identify school names:

Expression Name	Domain	Expression Level	Expression
School Name	SCHOOL_NM	Column	`(?>school_?na?me?)(?!\w*ID)`

Security Code

Below are the profile Expressions Delphix uses to identify security codes:

Expression Name	Domain	Expression Level	Expression
Security Code	SECURITY_CODE	Column	`(?>se?cu?r(i?ty?)?_?co?de?)(?!\w*ID)`

Serial Number

Below are the profile Expressions Delphix uses to identify serial numbers:

Expression Name	Domain	Expression Level	Expression
Serial Number	SERIAL_NM	Column	`(?>(ser(ial)?)_?(num(ber)?|nbr|no))(?!\w*ID)`

Signature

Below are the profile Expressions Delphix uses to identify signatures:

Expression Name	Domain	Expression Level	Expression
Signature	SIGNATURE	Column	`signature(?!\w*(ID|type))`

Social Security Number

Below are the profile Expressions Delphix uses to social security numbers:

Expression Name	Domain	Expression Level	Expression
Social Security Number	SSN	Column	`ssn(?!\w*ID)`
Data - SSN	SSN	Data	`\b(?!000)(?!666)[0-8]\d{2}[- ](?!00)\d{2}[- ](?!0000)\d{4}\b`

Tax ID

Below are the profile Expressions Delphix uses to identify tax IDs:

Expression Name	Domain	Expression Level	Expression
Tax ID Number	TAX_ID	Column	`tin$|^tin|_tin|tin_`
Tax ID Code or Number	TAX_ID	Column	`(ta?x)_?(id(ent)?)?_?((co?de?)|(num(ber)?|nbr|no))?`

Telephone Number

Below are the profile Expressions Delphix uses to identify telephone numbers:

Expression Name	Domain	Expression Level	Expression
Telphone or Contact Number	TELEPHONE_NO	Column	`(?>((tele?)?phone)|(co?nta?ct|tel)_?(num(ber)?|nbr|no))(?!\w*(ID|type))`
Data - Phone Number	TELEPHONE_NO	Data	`\(?\b[0-9]{3}\)?[-. ]?[0-9]{3}[-. ]?[0-9]{4}\b`
Fax Number	TELEPHONE_NO	Data	`(?>fax_?(num(ber)?|nbr|no)?)(?!\w*(ID|type))`

Vin Number

Below are the profile Expressions Delphix uses to identify vin numbers:

Expression Name	Domain	Expression Level	Expression
Vehicle	VIN_NO	Column	vehicle
VIN	VIN_NO	Column	`vin$|^vin|_vin|vin_`

Web Address

Below are the profile Expressions Delphix uses to identify web addresses:

Expression Name	Domain	Expression Level	Expression
Web or URL Address	WEB	Column	`(?>(url|web_?addre?s?s?))(?!\w*(ID|type))`
Data - Web Address	WEB	Data	`\b(?:(?:https?|ftp|file)://|www\.|ftp\.)[-A-Z0-9+&-@#/%=~_|$?!:,.]*[A-Z0-9+&-@#/%=~_\|$]`

ZIP Code

Below are the profile Expressions Delphix uses to identify zip codes:

Expression Name	Domain	Expression Level	Expression
zip or Postal Code	ZIP	Column	`(?>(zip|post(al)?)_?((co?de?)?4?))(?!\w*ID)`
Data - Zip Code	ZIP	Data	`1\b([0-9]{5})-([0-9]{4})\b`

Managing Domains

This section describes how you can create and manage your domains.

Domains specify certain data to be masked with a certain algorithm. From the **Settings** tab, if you click **Domains** to the left, the list of domains will be displayed. From here, you can add, edit, or delete domains.

Delphix Agile Data Masking includes several default domains and algorithms. These appear the first time you display the Masking Settings tab. Each domain has a classification and masking method assigned to it. You might choose to assign a different algorithm to a domain, but each domain name is unique and can only be associated with one algorithm. If you create additional algorithms, they will appear in the **Algorithms** drop-down menu. Because each algorithm you use must have a unique domain, you must add a domain (or reassign an existing domain) to use any other algorithms.

The **Domains** tab is where you define domains, along with their classification and the default Masking Algorithm.

Adding a New Domain

1. At the top of the **Domains** tab, click **Add Domain**.

2. Enter the new **Domain Name**. The domain name you specify will appear as a menu option on the **Inventory** screen elsewhere in the Delphix Masking Engine. Domain names must be unique.

3. Select the **Classification** (informational only). For example, customer-facing data, employee data, or company data.

4. Select a default **Masking Algorithm** for the new domain.

5. Click **Save**. To delete any domain, click the Delete icon to the far right of the domain name.

Configuring Profiling Settings

In addition to using your Rule Set to determine the inventory of what to profile, a Profiling job uses Expressions to identify your sensitive data. You can add regular expressions to be used by Profiler Sets to the Profiler Settings.

To display the Profiler Settings, click on the **Settings** tab and select **Profiler** on the left-hand side of the page.

The **Profiler Settings** screen displays Expressions along with their **Domain**, **Expression** text, Expression **Name**, **Owner**, and Expression profiling **Level**.

> ✏️ **Note**
>
> Column and data level expressions are case insensitive.

To add an Expression

1. Click **Add Expression** at the top of the Profiler screen.

Add Expression

Domain   Expression Name   Expression Level

[Select Domain   ▼] [   ] [Expression Level   ▼]

Expression Text

[   ]

_____

                                                    [Cancel]   [Submit]

1. Select a Domain from the **Domain** dropdown.

   - Domains are used by Profiling jobs to determine the masking Algorithm to apply to your sensitive data. When an Expression is matched, the Profiling job will associate the specified Domain to the sensitive data. The Masking Engine comes out of the box with over 30 pre-defined Domains. Domains can be added, edited, and deleted from the **Settings Domains** screen.

2. Enter the following information for the Expression:

   - **Expression Name**— The name used to select this expression as part of a Profiler Set.
   - **Expression Text**— The regular expression used to identify sensitive data.

3. Select an **Expression Level** for the Expression:

   - **Column Level**— To identify sensitive data based on column names.
   - **Data Level**— To identify sensitive data based on data values, not column names.

4. When you are finished, click **Save**.

To edit a saved Expression, click the **Edit** icon to the right of the Expression.

To delete an Expression

Click the **Delete** icon to the far right of the name.

Profiler Sets

Profiling jobs use Profiler Sets to determine the set of Expressions to use in identifying sensitive data in an Inventory. A Profiler Set is a grouping of Expressions for a particular purpose. For instance, First Name, Last Name, Address, Credit Card, SSN, and Bank Account Number Expressions could constitute a Financial Profiler Set.

The Masking Engine comes with two predefined Profiler Sets: Financial and Healthcare vertical. A Delphix Masking Engine administrator (a user with the appropriate role privileges) can create/add/update/delete these Profiler Sets.

If you want to edit or add a Profiler set, click **Profiler Set** at the top of the **Profiler Settings** screen. The Profiler Set dialog appears, listing the Profiler Sets along with their Purpose, Owner, and Date Created.

To add a Profiler Set

1. Click **Add Set** at the top of dialog window.

2. Enter a Profiler **Set Name**.

3. Optionally, enter a **Purpose** for this Profiler Set.

4. Enter or select which **Expressions** to include in this set.

5. When you are finished, click **Submit**.

To edit an existing Profiler Set, click the **Edit** icon to the right of the Profiler Set name.

To delete a Profiler Set

Click the **Delete** icon to the right of the Profiler Set name.

Creating A Profiling Job

This section describes how users can create a Profiling job. You can create Profiling jobs for databases, XML, copybooks, delimited files, and fixed-width.

The Profiler assigns each sensitive data element to a domain, with each domain having a default masking algorithm. Then, in the inventory, masking algorithms can be manually updated as needed to establish the masking rulesets for your data sources.

Profiling Jobs are grouped within environments on the **Environment Overview** page along with all masking jobs. In order to navigate to the **Overview** screen, click on an environment and the **Overview** tab should automatically display.

Creating a New Profiling Job

To create a new Profiling job:

1. Click the **Profile** button on the upper side of the page.

2. The **Create Profiling Job** window appears.

Create Profile Job

Job Name

Feedback Size

Target: Test

☐ **Multiple Profiler Expression Check**

☐ **Multi Tenant**

Profile Sets

Rule Set

-- Select Rule Set -- ▼

Profile Sets ▼

No. of Streams

Comments

1

Min Memory   **Max Memory**

In MB   In MB

Email

Cancel   Save

3. You will be prompted for the following information:

- **Job Name** — A free-form name for the job you are creating. Must be unique.

- **Multi Tenant** — Check the box if the job is for a multi-tenant database. This option allows existing rulesets to be re-used to mask identical schemas via different connectors. The connector is selected at job execution time.

- **Rule Set** — Select the rule set that this job will profile.

- **No. of Streams** — The number of parallel streams to use when running the jobs. For example, you can select two streams to profile two tables in the ruleset concurrently in the job instead of one table at a time.

- **Min Memory (MB)** — (optional) Minimum amount of memory to allocate for the job, in megabytes.

- **Max Memory (MB)** — (optional) Maximum amount of memory to allocate for the job, in megabytes.

- **Feedback Size** — (optional) The number of rows to process before writing a message to the logs. Set this parameter to the appropriate level of detail required for monitoring your job. For example, if you set this number significantly higher than the actual number of rows in a job, the progress for that job will only show 0 or 100%.

- **Multiple Profiler Expression Check** - By default, the profiler stops testing Profiler Expressions on a column or data value after the first expression matches. Check this box if the job should check all Profiler Expressions. If multiple Profiler Expressions match, the Profiler report will indicate multiple matches and the algorithm specified by the `DefaultMultiphiAlgorithm` application setting will be assigned.

- **Profile Sets** — The name of the Profile Set to use. A Profile Set is a set of Profile Expressions (for example, a set of financial expressions).

- **Comments** — (optional) Add comments related to this job.

- **Email** — (optional) Add e-mail address(es) to which to send status messages. Separate addresses with a comma (,).

4. When you are finished, click **Save**.

Running A Profiling Job

This section describes how users can run a profiling job from the **Environment Overview** screen.

To run or rerun a job from the **Environment Overview** screen:

- Click the **Run** icon (play icon) in the Action column for the desired job.

- The **Run** icon changes to a **Stop** icon while the job is running.

- When the job is complete, the **Status** changes.

To stop a running job from the **Environment Overview** screen:

1. Locate the job you want to stop.

2. In the job's **Action** column, click the **Stop** icon.

3. A popup appears asking, "Are you sure you want to stop job?" Click **OK**.

   When the job has been stopped, its status changes.

Reporting Profiling Results

This section describes the different ways of sharing/exploring the results of a Profiling job.

After a Job has been started from the Environment **Overview** screen, clicking on the Job Name will result in the display of the Profiling job from the **Monitor** tab. Clicking on the **Results** tab in the middle of the screen after the job has completed will display the sensitive data findings on a table-column by table-column or file-field by file-field basis.

To retrieve a PDF report of the **Results** tab, click on the **Profiling Report** link near the top of the page.

Profiling Report

Job Profile		Job Status	
Name	MSSQLServer	Current Status :	Succeeded
Type :	Profiling	Start Time :	09/10/18 21:03
Environment :	Test	End Time :	09/10/18 21:03

Schema	Table	Column	Domain	Algorithm
dbo	Foo1	fname	FIRST_NAME	FIRST NAME SL
dbo	Foo1	lname	LAST_NAME	LAST NAME SL
dbo	Foo2	fname	FIRST_NAME	FIRST NAME SL
dbo	Foo2	lname	LAST_NAME	LAST NAME SL

Alternatively, after a job completes successfully, the profiling results can be displayed through the **Inventory** screen by examining the assigned **Domain** and masking algorithm **Method**s for tables/files in the Rule Set.

To get a spreadsheet capturing the Profiling results for the inventory, click on **Export** near the top of the page and a CSV file will be created.

MSSQL_Server_10917744884

Environment Name	Rule Set	Table Name	Type	Parent Column Name	Column Name	Data Type	Domain	Algorithm	Is Masked
Test	MSSQL Server	TEST4TABLE	-	-	TT1_COL1_DOT	varchar (100)	-	-	false
Test	MSSQL Server	TEST4TABLE	-	-	TT1_COL2_DOT	varchar (100)	-	-	false
Test	MSSQL Server	TEST4TABLE	-	-	ID1_DOT	int (0)	-	-	false
Test	MSSQL Server	Foo	-	-	IDD	int (0)	-	-	false
Test	MSSQL Server	Foo1	-	-	lname	varchar (50)	LAST_NAME	LAST NAME SL	true
Test	MSSQL Server	Foo1	-	-	fname	varchar (50)	FIRST_NAME	FIRST NAME SL	true
Test	MSSQL Server	Foo1	PK ID IX	-	idd	int (0)	-	-	false
Test	MSSQL Server	Foo2	PK ID IX	-	idd	int (0)	-	-	false
Test	MSSQL Server	Foo2	-	-	fname	varchar (50)	FIRST_NAME	FIRST NAME SL	true
Test	MSSQL Server	Foo2	-	-	lname	varchar (50)	LAST_NAME	LAST NAME SL	true

The spreadsheet can then be shared and manually modified to correct the sensitive data findings by:

1. Changing the **Is Masked**, **Algorithm**, and/or **Domains** fields for the respective Table/Column or File/Field in the CSV file accordingly.

2. Importing the modified spreadsheet by clicking on **Import** near the top of the **Inventory** screen and specifying the modified CSV file name.

Securing Sensitive Data

Algorithms

Introduction to Masking Algorithms

This section provides a brief overview of the different algorithm options that are available and other general algorithm information.

Algorithm Options

Out Of The Box Algorithm Instances

Out of the box algorithm instances are pre-configured ready to use algorithms. The out of the box algorithms with related frameworks can be customized using the corresponding extensible frameworks. For more information on algorithm instance extensibility, see Extensible Algorithms.

Algorithm Instances	Extensible?	Related Framework
dlpx-core:CM Alpha-Numeric	X	Character Mapping
dlpx-core:CM Digits	X	Character Mapping
dlpx-core:CM Numeric	X	
Credit Card	X	Payment Card
Date Shift Discrete	X	
Date Shift Fixed	X	Date Shift
Date Shift Variable	X	

Algorithm Instances	Extensible?	Related Framework
dlpx-core:Email SL	X	Email
dlpx-core:Email Unique	X	Email
dlpx-core:LastName	X	Name
dlpx-core:FirstName	X	Name
dlpx-core:FullName	X	Full Name
SecureShuffle	X	

Algorithm Frameworks

Algorithm frameworks allow for creation of algorithm instances with a custom configuration. For more information on algorithm framework extensibility, see Extensible Algorithms. More information on multi-column algorithms can be found at Using Multi-Column Algorithms.

Algorithm Framework	Extensible?	Multi-Column?	Out of the Box Instances
Binary Lookup			
Character Mapping	X		dlpx-core:CM Alpha-Numeric dlpx-core:CM Digits
Data Cleansing			
Date Replacement	X		
Date Shift	X		Date Shift Fixed
Dependent Date Shift	X	X	
Email	X		dlpx-core:Email Unique dlpx-core:Email SL
Free Text Redaction			
Full Name	X		dlpx-core:FullName

Algorithm Framework	Extensible?	Multi-Column?	Out of the Box Instances
Mapping			
Min Max			
Name	X		dlpx-core:FirstName dlpx-core:LastName
Payment Card	X		Credit Card
Regex Decompose	X		
Secure Lookup	X		
Tokenization			

Configuring Your Own Algorithms

Algorithm Settings

The **Algorithm** tab displays algorithm Names along with Type and Description. This is where you add (create) new algorithms. The default algorithms and any algorithms you have defined appear on this tab.

At the top of the page, **Nonconforming Data behavior** is displayed to specify how all algorithms should behave if they encounter data values in an unexpected format. **Mark job as Failed** instructs algorithms to throw an exception that will result in the job failing. **Mark job as Succeeded** instructs algorithms to ignore the non-conformant data and not throw an exception. Note that **Mark job as Succeeded** will result in the non-conformant data not being masked should the job succeed, but the **Monitor** page will display a warning that can be used to report the non-conformant data events.

Creating New Algorithms

If none of the default algorithms meet your needs, you might want to create a new algorithm. An algorithm that you create is called a "user-defined algorithm".

Algorithm Frameworks give you the ability to quickly and easily define the algorithms you want, directly on the Settings page. After you create an algorithm, your algorithm will be available to all users.

To add an algorithm:

1. In the upper right-hand corner of the **Algorithm** settings tab, click **Add Algorithm**.

2. Select an algorithm type.

3. Complete the form to the right to name and describe your new algorithm.

4. Click **Save.**

Editing Algorithms

Administrators can update **system**-defined algorithms. User-defined algorithms can be updated by the owner/user who created the algorithm.

Algorithm Frameworks Overview

Choosing an Algorithm Framework

See the Algorithm Frameworks section for a detailed description of each Algorithm Framework. The algorithm framework you choose will depend on the format of the data and your internal data security guidelines.

Choosing Between Character and Segment Mapping Frameworks

The Character Mapping algorithm is intended to replace Segment Mapping for many use cases. That said, it does not replicate every feature of that algorithm, so the specific masking application will determine which one is appropriate.

Reasons to choose Character Mapping over Segment Mapping:

- Character Mapping has no limit on the number of positions masked. Segment Mapping cannot handle inputs longer than 36 maskable characters.

- Character Mapping can mask all characters in the first Unicode plane. Segment Mapping can only mask "[a-zA-Z]" + "[0-9]"

- Character Mapping automatically preserves all non-masked characters. Segment Mapping requires configuration of all preserve characters, which can be impossible due to the limit on the number of preserve characters. Character Mapping is much easier to use when the data is potentially "dirty" or not consistently formatted.

- Character Mapping always changes the input (unless no maskable characters are present). With Segment Mapping, there is typically a small chance an input will mask to the same value.

- Character Mapping can process preserve ranges in reverse, allowing the last positions of an input to be preserved when inputs have different lengths. Segment Mapping preserve ranges are always processed from the beginning of input.

- Character Mapping uses a more complex masking computation, so that every maskable position influences every other position in the masked value. Segment Mapping pre-computes the permutations for each segment independently.

Reasons to choose Segment Mapping over Character Mapping:

- Segment mapping can mask different parts of the input, determined by position, differently. Character Mapping always masks the same groups of characters regardless of position.

- Segment mapping can map inputs to different outputs at a position, like { A, B, C, D } -> { W, X, Y, Z } by specifying different *Real* and *Mask* values. This is not possible with Character Mapping.

- Segment mapping supports numeric segments, with up to 4-digit segments masked to a specific range. Character Mapping doesn't allow this kind of range limiting.

- Segment Mapping can be used for tokenization. Character Mapping does not support tokenization at this time.

Out Of The Box Algorithm Instances

dlpx-core:CM Alpha-Numeric

> **Based on Extensible Algorithm Framework**

The CM Alpha-Numeric algorithm is an instance of the Character Mapping Algorithm Framework.

This algorithm masks all ASCII digit, lowercase, and uppercase characters, as well as some extended latin and cyrillic characters. Refer to the framework description for details of how masking is performed.

At least one character in the input must be masked, or Non-Conformant data handling will be triggered.

For example:

- "6379315274824970" → "0345698341375224"

- "ABCxyz123" → "HANwhp391"

- "Sí" → "Cž"

- "999-12-3456." → "668-23-1138."

- "2000:a86f::1" → "3893:u55x::0"

> ✎ **Note**
>
> This algorithm may generate non-conformant data events.

dlpx-core:CM Digits

Based on Extensible Algorithm Framework

The CM Digits algorithm is an instance of the Character Mapping Algorithm Framework.

This algorithm masks all ASCII digits. Refer to the framework description for details of how masking is performed. Be aware that this algorithm can produce value collisions when applied to Numeric data types. This is because leading zeros are not significant in numeric types, so while "7" → "8" and "304" → "008" may be different string results, when inserted into a numeric field, they represent the same value. If this behavior is undesirable, consider using the CM Numeric algorithm.

At least one character in the input must be masked, or Non-Conformant data handling will be triggered.

For example:

- "6379315274824970" → "8345698341375224"

- "99" → "05"

- "ABCxyz123" → "ABCxyz391"

- "0" → "6"

> ✏️ **Note**
>
> This algorithm may generate non-conformant data events.

dlpx-core:CM Numeric

Based on Extensible Algorithm Framework

The CM Numeric algorithm is an algorithm based on logic in the Character Mapping Algorithm Framework.

The framework this algorithm is based on is not configurable and cannot be reused to create additional instances.

This algorithm masks all ASCII digit without the possibility of the first digit masking to "0". Leading and trailing zeros are preserved. The value "0" always masks to "0". Unlike the "CM digits" instance, the number of significant digits is always preserved for all numeric inputs.

Refer to the framework description for details of how masking is performed.

At least one character in the input must be masked, or Non-Conformant data handling will be triggered.

For example:

- "6379315274824970" → "5210366768740261"

- "99" → "75"

- "000051.1230" → "000072.9040"

- "ABCxyz123" → "ABCxyz391"

- "0" → "0"

> ✏️ **Note**
>
> This algorithm may generate non-conformant data events.

Credit Card

Based on Extensible Algorithm Framework

The Credit Card algorithm is an instance of the Payment Card Algorithm Framework. The algorithm requires input values to have at least 8 digits in the character group [0-9]. If an input value has less than this, the algorithm will return an error. It preserves the first 6 digits of the input and requires at least one position to be masked for masking to be considered successful. The algorithm masks all subsequent digits by replacing them with a random value. All input characters that are not in the character group [0-9] are preserved. The algorithm maintains Luhn check validity through masking so input values with a valid Luhn check will mask to a value with a valid Luhn check. The out-of-the-box instance of this algorithm is called **CreditCard**.

For example:

- "6379315274824970" → "6379318341375224"

- "6379.3152.7482.4970" → "6379.3183.4137.5224"

- "abc5473defg04828hijkl0656253" → "abc5473defg04971hijkl6490341"

> ✎ **Note**
>
> This algorithm may generate non-conformant data events.

Date Shift Discrete

The Date Shift Discrete algorithm masks all dates with the same year-month combination to the same day. A different day is returned for each year-month combination. As an example, any inputs with a year-month combination of February 2020 may return a day value of 23 while any inputs with a year-month combination of January 2020 may return a day value of 5. All values of the input other than the day value are preserved. This algorithm is deterministic based on an algorithm key. The out-of-the-box instance of this algorithm is called **DateShiftDiscrete**.

For example:

- "1989-11-19 00:00:00" → "1989-11-30 00:00:00"
- "1989-12-19 04:15:00" → "1989-12-24 04:15:00"
- "2012-11-19 17:00:55" → "2012-11-08 17:00:55"
- "2012-11-09 00:23:59" → "2012-11-08 00:23:59"

> ✏️ **Note**
>
> This algorithm may generate non-conformant data events.

Date Shift Fixed

> **Based on Extensible Algorithm Framework**

The Date Shift Fixed algorithm is an instance of the Date Shift Algorithm Framework masking the input to 5 days in the future with roll enabled so only the day of the month will change, all other units will remain the same. Dates at the end of the month will roll back to the beginning of the same month in the same year. The out-of-the-box instance of this algorithm is called **DateShiftFixed**.

For example:

- "2001-02-05 12:30:00" → "2001-02-10 12:30:00"
- "2001-02-27 15:45:00" → "2001-02-04 15:45:00"
- "2001-12-28 00:00:00" → "2001-12-02 00:00:00"

> ✏️ **Note**
>
> This algorithm may generate non-conformant data events.

Date Shift Variable

The Date Shift Variable algorithm returns a random date within the same month-year as the input date. Dates will not mask to the original input date. This algorithm may produce collisions. The out-of-the-box instance of this algorithm is called **DateShiftVariable**.

For example:

- "2019-02-05 10:00:00" → "2019-02-13 10:00:00"
- "2019-02-12 15:30:00" → "2019-02-13 15:30:00"
- "2019-02-27 00:45:30" → "2019-02-17 00:45:30"
- "2020-02-27 00:00:00" → "2020-02-22 00:00:00"

> ✏ **Note**
>
> This algorithm may generate non-conformant data events.

dlpx-core:Email SL

Based on Extensible Algorithm Framework

The Email SL algorithm is an instance of the Email Algorithm Framework. This algorithm splits the input on the '@' symbol. Handling of malformed inputs is detailed on the Email Algorithm Framework page. This algorithm does not generate any non-conformant data events. The algorithm will split the input into two parts: **name** and **domain**. Name is the portion before the '@' symbol and domain is the portion after the '@' symbol.

A secure lookup is applied to the name portion of the input. The provided secure lookup file contains 20,000 unique lookup values in various formats. The following formats are used in the default lookup file:

- FirstName.LastName

- FirstName_LastName

- FirstInitial.LastName

- FirstNameLastName

- FirstNameLastInitialNumber

The domain portion is replaced by the fixed value "example.com". This value is a reserved domain with a valid DNS entry.

This algorithm is deterministic based on an algorithm key. It is possible that there may be collisions where two different values mask to the same value due to the nature of secure lookup. The out-of-the-box instance of this algorithm is called **dlpx-core:Email SL**.

For example:

- "bob@gmail.com" → "E.Duboise@example.com"

- "bob@hotmail.com" → "E.Duboise@example.com"

- "alex@gmail.com" → "OrvinA436@example.com"

- "joe_123@yahoo.com" → "Amil.Steidinger@example.com"

dlpx-core:Email Unique

> **Based on Extensible Algorithm Framework**

The Email Unique algorithm is an instance of the Email Algorithm Framework. This algorithm splits the input on the '@' symbol. Handling of malformed inputs is detailed on the Email Algorithm Framework page. This algorithm does not generate any non-conformant data events. The algorithm will split the input into two parts: **name** and **domain**. Name is the portion before the '@' symbol and domain is the portion after the '@' symbol.

The name portion is masked by performing a SHA-256 hash of the entire input (including the domain). This means that inputs with the same name portion but different domain portions will mask to different values. The hashed value is then encoded using Base32 encoding. The result of these transformations is the masked name portion.

The domain portion is replaced by the fixed value "example.com". This value is a reserved domain with a valid DNS entry.

This algorithm is deterministic based on an algorithm key. This algorithm provides unique masked values for each input. The out-of-the-box instance of this algorithm is called **dlpx-core:Email Unique**.

For example:

- "bob@gmail.com" → "XF35TNMKPPTMQF4CX5264ZRXOMJJL2DQVE3KTZNIJ2NS6EUH7GLA@example.com"
- "bob@hotmail.com" → "M2U3LCC24MP5XDQ7DH4RSDW6QXCWRTSJVQF22C7IKBXDQ3LBM7NQ@example.com"
- "alex@gmail.com" → "CQKOXVBPD3VT42XHLBBUHEWIAJ26X3NROEBZHMSC7B4NFSZSTBIQ@example.com"
- "joe_123@yahoo.com" → "JTJNSLWLK4TWQ7VKG2KMRMMMH4M3FRIXUXFR7TIEL6VJR3G6AU2Q@example.com"

dlpx-core:FirstName

| **Based on Extensible Algorithm Framework**

The First Name algorithm is an instance of the Name Algorithm Framework. The algorithm requires String type input values.

The expected format for the valid input contains at least one word, which consist of any symbol(s), but the single non-alphanumeric character. Any single non-alphanumeric character would be ignored, replaced with empty string "". If the input value does not match the expected format, the value will not be masked.
For example
- single punctuation mark is considered as non-conformant data and is not masked (empty string is returned instead).
- if input contains null or empty string or white spaces only then the algorithm returns unmasked input value.

Word containing any multiple characters (even non-alphanumeric) is considered as a valid input and is masked.
No non-conformant data errors are thrown by that algorithm.

Single character (if alphanumeric only) is considered abbreviation. Whether it is followed by the dot (.) ot not.

Words separated by the hyphen (-) are considered as a single word (even if divided from hyphen by spaces).

The default First Name instance is configurable without particle files. So every input word (but the mentioned above single non-alphanumeric symbol) is considered as a valid part of the name. The whole input would be masked to a single output word. Leading and trailing white spaces are not preserved.

For example:

Input	Masked Output
null	null
"" (empty string)	""
" " (white spaces only)	" "
single non alphanumeric character (like '&' or '?')	""
&?	Michael
M	S
M.	S.
Ann- Marie	Boris

Input	Masked Output
von (particle)	Tim
Eric Maria	Kurt

dlpx-core:FullName

> **Based on Extensible Algorithm Framework**

The Full Name algorithm is an instance of the Full Name Algorithm Framework. The algorithm requires String type input values.

If input value is non-conformant - it's not masked.
For example - single punctuation mark is considered as non-conformant data and is not masked (empty string is returned instead).
But word containing any multiple characters is considered as a valid input and masked.
Words separated by hyphen (-) are considered as a single word (even if divided from hyphen by spaces).
No non-conformant data errors are thrown by that algorithm.

The default Full Name algorithm instance uses all default parameters, and chains "dlpx-core:FirstName" algorithm instance for first names masking, and "dlpx-core:LastName" for last name masking.

Below are few examples of the Full Name default algorithm instance masking:

Input	Masked Output
Manuel Maria Saxe-Coburgo-Gotha	Nimisha Kum Mcneish
Manuel - Boris Maria Saxe-Coburgo-Gotha	Simeon Kum Mcneish
Manuel Maria Saxe -Coburgo - Gotha	Nimisha Kum Mcneish
Manuel Maria de Saxe-Coburgo-Gotha	Nimisha Kum Mcneish
Manuel Maria de Saxe-Coburgo-Gotha (*)	Nimisha Kum Casteleyn
Manuel Maria - de ? Saxe-Coburgo-Gotha : #	Nimisha Muharrem Mcneish
Manuel Maria Saxe-Coburgo-Gotha (*)	Nimisha Kum Casteleyn
Mr. Manuel Maria de Saxe-Coburgo-Gotha #	Nimisha Kum Mcneish

Input	Masked Output
Saxe-Coburgo-Gotha, Manuel Maria	Mcneish, Nimisha Kum
saxe-coburgo-gotha, Manuel Maria	mcneish, Nimisha Kum
SAXE-COBURGO-GOTHA, MANUEL Maria	MCNEISH, NIMISHA Kum
Saxe-Coburgo-Gotha: M. M	Claudia T. S
M. G. Maria Saxe-Coburgo-Gotha	T. E. Mcneish
M M Saxe-Coburgo-Gotha	T T Mcneish
M M. Saxe-Coburgo-Gotha	T T. Mcneish
M M. S	T T. G
M M. S.	T T. G.
m m. s.	t t. g.
Max	Grassi
Max	Grassi

dlpx-core:LastName

Based on Extensible Algorithm Framework

The Last Name algorithm is an instance of the Name Algorithm Framework. The algorithm requires String type input values.

The expected format for the valid input contains at least one word, which consist of any symbol(s), but the single non-alphanumeric character. Any single non-alphanumeric character would be ignored, replaced with empty string "". If the input value does not match the expected format, the value will not be masked.
For example
- single punctuation mark is considered as non-conformant data and is not masked (empty string is returned instead).
- if input contains null or empty string or white spaces only then the algorithm returns unmasked input value.

Word containing any multiple characters (even non-alphanumeric) is considered as a valid input and is masked.
No non-conformant data errors are thrown by that algorithm.

Input containing multiple words is masked to a single word (after configured particles are removed from the input). Single word input is not checked for configured particles. Single character (if alphanumeric only) is considered abbreviation. Whether it is followed by the dot (.) ot not.
Words separated by hyphen (-) are considered as a single word (even if divided from hyphen by spaces).

The default Last Name instance is configurable with particleToRemove file. The whole input would be masked to a single output word. Leading and trailing white spaces are not preserved.

For exmaple:

Input	Masked Output
null	null
"" (empty string)	""
" " (white spaces only)	" "
single non alphanumeric character (like '&' or '?')	null
M	S
M.	S.
?>	Michael
Ann- Marie	Boris

Input	Masked Output
von (particle)	Wilke
Lister Weissman	Vonk
Frout	Smith
von Frout	Smith

Particles treatment:

Input	Masked Output	configuration
dela Cruz	dela Lordello	particle "dela" is configured to be preserved
dela Cruz	Lordello	particle "dela" is configured to be removed
dela Cruz	Lordello	particle "dela" is configured in both - toPreserve and toRemove lists
dela Cruz	Santos	particle "dela" isn't listed in any particles list

SecureShuffle

This algorithm masks by shuffling the values is a particular field or column to different lines or rows. For example, values for the FIRST_NAME column might be shuffled among a number database rows within a table. It guarantees that each value is moved to a different line or row, but will not prevent an input from masking to the same output in the case where the values shuffled are not unique.

> **ℹ Info**
>
> Because shuffling data does not redact or modify the individual data values in any way, careful consideration must be given to whether this form of obfuscation is sufficient to meet your security requirements.

The SecureShuffle algorithm may only be used with masking jobs that support batching, and will not be presented as an option in the inventory screen when it is not supported. The maximum number positions any particular value will be moved within the input is equal to the batch size.

Please refer to the Batch Masking section here for a full description of the Batch Masking mechanism, as well as details on batch size and which jobs support batching.
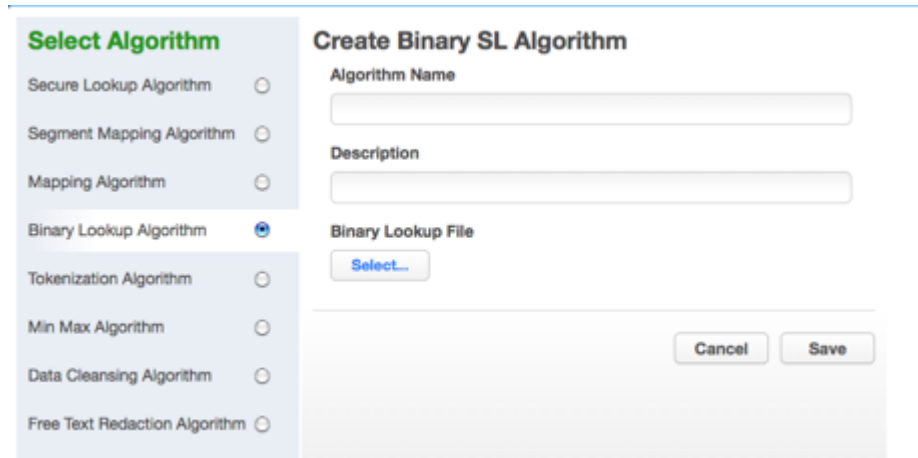
This algorithm will report non-conformant data whenever only one value is available to mask, meaning that no shuffling is possible.

Algorithm Frameworks

Binary Lookup

A Binary Lookup algorithm is much like the Secure Lookup algorithm but is used when entire files are stored in a specific column. This algorithm replaces objects that appear in object columns. For example, if a bank has an object column that stores images of checks, you can use a Binary Lookup algorithm to mask those images. The Delphix Engine cannot change data within images themselves, such as the names on X-rays or driver's licenses. However, you can replace all such images with a new, fictional image. This fictional image is provided by the owner of the original data.

Creating a Binary Lookup Algorithm via UI

1. At the top right of the **Algorithm** tab, click **Add Algorithm**.

2. Select **Binary Lookup Algorithm**. The Create Binary SL Algorithm pane appears.

3. Enter an **Algorithm Name**.

> **ℹ Info**
>
> This MUST be unique.

4. Enter a **Description**.

5. Select a **Binary Lookup File** on your filesystem.

6. Click **Save**.

For information on creating Binary Lookup algorithms through the API, see API Calls for Creating Algorithms - Binary Lookup.

Character Mapping

> **Extensible Algorithm Framework**

The Character Mapping framework maps text values, defined by a set of character groups, to other text values generated from the same character groups. Mappings are calculated algorithmically, so it is not necessary to provide the set of mapping values. The algorithm preserves any characters not assigned to a group. Any characters from the first Unicode plane can be mapped - this covers most characters used in modern languages. Other (supplementary) characters can only be preserved.

The particular set of permutations used is determined by the algorithm's key, so rekeying the algorithm will cause different outputs to be generated for each input.

The algorithm has the following properties:

- The masked value for each input is consistent unless the algorithm is rekeyed.

- No two text inputs produce the same text output. Collisions *are* possible for some data types, such as Numeric, where multiple text values, such as "001" and "1", are treated as the same value.

- As long as at least one maskable character is present in the input, the masked value will never match the input.

- Each masked position influences the mapping done at every other masked position.

For these reasons, this algorithm is useful for masking columns with uniqueness requirements, such as primary and foreign key columns.

This algorithm was introduced in version 6.0.5.0, and uses the algorithm extensibility framework, allowing it to be called from other algorithms using that framework.

To decide whether Character Mapping or Segment Mapping is the correct option for your use case, see Choosing Between Character and Segment Mapping Frameworks.

Creating a Character Mapping Algorithm via UI

Select Framework

Secure Lookup ○

Character Mapping ◉

Segment Mapping (legacy) ○

Mapping ○

Binary Lookup ○

Tokenization ○

Min Max ○

Data Cleansing ○

Free Text Redaction ○

Create Character Mapping Algorithm

Algorithm Name

Description

Character Groups ❓ Learn More

| Select group | **Add** |

☐ **Case Sensitive**

Minimum Masked Positions

1 ▲▼

☐ **Preserve Leading Zeros**

Preserve Ranges

Starting Position   Length   Direction

[   ]   [   ]   Forward ▼   **Add**

Cancel   **Save**

1. In the upper right-hand region of the **Algorithm** tab under **Settings**, click **Add Algorithm**.

2. Select **Character Mapping Algorithm**. The "Create Character Mapping Algorithm" pane appears.

3. Enter an **Algorithm Name**.

> ℹ **Info**
>
> This MUST be unique.

4. Enter a **Description.**

5. Define **Character Groups** for each group of characters among which you would like to map. Each group may defined either by specifying each literal character in the group, such as "0123456789", or using Java Regular Expression style character ranges, such as "[0-9]". The algorithm will freely map characters to other characters within the same group, so by defining groups "[0-9]" and "[A-Z]", numbers would be replaced by other numbers, and letters by other letters, but a number would never be replaced by a letter. Groups should not contain duplicate characters, and each character may belong to only one group. Any character that is not assigned to a group will be preserved (not masked) by the algorithm.

   The box below the entry area allows selection of character groups defined for other, preexisting Character Mapping algorithms.

6. Check the **Case Sensitive** box to cause the algorithm to treat upper and lower case characters as distinct characters for mapping.

7. Select a value for **Minimum Masked Position**, which sets the minimum number of characters that the algorithm must mask; fewer positions triggers non-conformant data handling. Null, empty, and all-whitespace values never trigger non-conformant data handling.

8. Check the **Preserve Leading Zeros** box to cause the algorithm to preserve any number of '0' characters at the beginning of each input. This is only useful if '0' has been assigned to a character group in step 5.

9. If desired, define ranges of the input value to ignore using the **Preserve Ranges** controls. For Character Mapping algorithms, only characters that would otherwise be masked count when determining position for preserve ranges. Each preserve range is defined by:

   - **Start Position** - The position at which to start preserving, starting from 0.

   - **Length** - The number of characters to preserve.

   - **Direction** - The direction, either forward or reverse, determining whether to process from the beginning or end of input for this range.

> ⚠️ **Warning**
>
> Be wary of the following Preserve Range processing differences between Segment Mapping and Character Mapping: Segment Mapping ranges start with index 1, while Character Mapping ranges begin with index 0. Segment Mapping includes perserved characters when determining position, while Character Mapping only counts maskable characters. For example, to ignore the first two characters, you would enter Starting Position 0 for Character Mapping, but Starting Position 1 for Segment Mapping. If both algorithms were configured to preserve "-", and preserve the first two positions, Character Mapping might mask "--0000" to "--0073", while Segment Mapping might mask "--0000" to "--4638".

Examples

As an example, a Character Mapping algorithm could be defined with a single character group, "[0-9]". It might mask as follows:

- "(603) 867-5309" → "(463) 638-0193"

- "999-12-3456" → "453-71-6283"

- "Call Tom at 8:00PM" → "Call Tom at 2:75PM"

- "Call Tom at 8:00PM" → "Call Tom at 2:75PM"

Data Cleansing

A data cleansing algorithm does not perform any masking. Instead, it standardizes varied spellings, misspellings, and abbreviations for the same name. For example, "Ariz," "Az," and "Arizona" can all be cleansed to "AZ." Use this algorithm if the target data needs to be in a standard format prior to masking.

Creating a Data Cleansing Algorithm via UI

1. Enter an **Algorithm Name**.

> **ⓘ Info**
>
> This MUST be unique.

2. Enter a **Description**.

3. Select **Lookup File** location.

4. Specify a **Delimiter** (key and value separator). The default delimiter is =. You can change this to match the lookup file.

5. Click **Save**.

Below is an example of a lookup input file. It does not require a header. Make sure there are no spaces or returns at the end of the last line in the file. The following is sample file content:

```
NYC=NY
NY City=NY
New York=NY
Manhattan=NY
```

For information on creating Data Cleansing algorithms through the API, see API Calls for Creating Algorithms - Data Cleansing.

Date Replacement

> **Extensible Algorithm Framework**

The Date Replacement framework masks a date value based on specified beginning and end dates. Masked output values are calculated algorithmically using the algorithm's key, so rekeying the algorithm will cause a different output value to be generated for each input. It is possible for an input to be masked to itself.

Creating a Date Replacement Algorithm via UI

Select Framework

Secure Lookup	○
Character Mapping	○
Payment Card	○
Date	●
Dependent Date Shift	○
Segment Mapping (legacy)	○
Mapping	○
Binary Lookup	○
Tokenization	○
Min Max	○
Data Cleansing	○
Free Text Redaction	○

Create Date Algorithm

Algorithm Name

[   ]

Description

[   ]

Select Algorithm type   ❓ Learn More

● **Replacement:**   [Min Date 📅🕐]   [Max Date 📅🕐]

○ **Shift:**   [Min Value]   [Max Value]

Unit

[SECONDS ▼]

[Cancel]   [Save]

1. In the upper right-hand region of the **Algorithm** tab under **Settings**, click **Add Algorithm**.

2. Select **Date**. The "Create Date Algorithm" pane appears.

3. Enter an **Algorithm Name**.

> ℹ️ **Info**
>
> This MUST be unique.

4. Enter a **Description**.

5. Under **Select Algorithm type** choose **Replacement**.

6. Enter **Min Date** and **Max Date**. These define the range from which the algorithm will select output values. The range is inclusive of both values. All units of time less than the specified unit must be set to 0. For example, a configuration with the unit set to **Days** must have the time portion set to 00:00:00.

7. Choose the **Unit** of time form the drop-down: **Days**, **Hours**, **Minutes**, or **Seconds**. This represents the unit of time the range is expressed in. Any unit smaller than the specified unit will be set to 0 in the masked output. For example, with a unit of **Days**, all masked time values will be 00:00:00. For a more detailed explanation, see the Examples section.

8. When you are finished, click **Save**.

For information on creating Date Replacement algorithms through the API, see API Calls for Creating Algorithms - Date Replacement.

Examples

As an example, a Date Replacement algorithm with a minimum range of "2020-01-01 00:00:00" and a maximum range of "2020-01-05 00:00:00" with the unit set to **Days** will replace the input value with a date in the specified range. Dates may mask as follows:

- "1995-03-05 13:25:00" → "2020-01-02 00:00:00"

- "2021-10-13 01:59:59" → "2020-01-04 00:00:00"

- "1856-07-31 00:00:00" → "2020-01-01 00:00:00"

Another example with a minimum range of "2020-01-01 01:00:00" and a maximum range of "2020-01-01 03:00:00" with the unit set to **Hours** provides 3 possible mask values:

- "2020-01-01 01:00:00"

- "2020-01-01 02:00:00"

- "2020-01-01 03:00:00"

Using the same range of "2020-01-01 01:00:00" to "2020-01-01 03:00:00" but with the unit set to **Minutes**, there are 121 possible output values as the unit is the granularity at which time is subdivided. Note that the range is inclusive of both range values. Possible masked values may be as follows:

- "2020-01-01 01:00:00"

- "2020-01-01 01:14:00"

- "2020-01-01 01:59:00"

- "2020-01-01 02:23:00"

- "2020-01-01 03:00:00"

All inputs with the same value masked with the same algorithm configuration will result in the same output values.

Date Shift

| **Extensible Algorithm Framework**

The Date Shift framework masks date values to different dates based on a specified range around the input value. Masked values are calculated algorithmically using the algorithm's key, so rekeying the algorithm will cause different outputs to be generated for each input. All valid input values will be masked to a new value, and the new value will never match the input.

Creating a Date Shift Algorithm via UI

1. In the upper right-hand region of the **Algorithm** tab under **Settings**, click **Add Algorithm**.

2. Select **Date**. The "Create Date Algorithm" pane appears.

3. Enter an **Algorithm Name**.

> **ⓘ Info**
>
> This MUST be unique.

4. Enter a **Description**.

5. Under **Select Algorithm type** choose **Shift**.

6. Enter **Min Value** and **Max Value**. These values provide a range in which the masked value will differ from the input given a specified unit of time. The range is inclusive of both values where negative values represent units of time in the past and positive values represent units of time in the future. 0 may be included in the range or as one of the range values, but the input will not mask to the same value. A minimum value and maximum value that are equal will result in a fixed shift of that amount of time. For example, entering 3 as a min value and 3 as a max value with a unit of **Days** will mask all input values to 3 days in the future.

7. Check the **Roll** box to preserve all units of time larger and smaller than the specified unit. Only the value of the specified unit will change. This option is supported for units months, days, hours, minutes, and seconds.

8. Choose the **Unit** of time from the drop-down: **Years**, **Months**, **Days**, **Hours**, **Minutes**, or **Seconds**. This represents the unit of time the range is expressed in.

9. When you are finished, click **Save**.

For information on creating Date Shift algorithms through the API, see API Calls for Creating Algorithms - Date Shift.

Examples

As an example, a Date Shift algorithm with a minimum value of 3 and a maximum value of 5 with the unit set to **Days** will shift the input value from 3 to 5 days into the future. Dates may mask as follows:

- "2021-02-03 12:30:00" → "2021-02-06 12:30:00"

- "1905-12-10 00:00:00" → "1905-12-15 00:00:00"

- "2001-07-31 23:45:30" → "2001-08-04 23:45:30"

With roll enabled and the same configuration, a date at the end of a month will wrap around to the beginning of the month. Dates may mask as follows:

- "2021-02-25 10:00:00" → "2021-02-01 10:00:00"

- "1932-05-03 01:15:15" → "1932-05-08 01:15:15"

- "1999-08-31 18:30:00" → "1999-08-03 18:30:00"

All inputs with the same value masked with the same algorithm configuration will result in the same output values.

Dependent Date Shift

> **Extensible Algorithm Framework**

The Dependent Date Shift algorithm provides a method to manipulate dates together where a depdendency exists between the two dates that must be maintained. Examples of this include date of admission and date of discharge or date of birth and date of death. If we were to attempt to mask these dates indepdendently, we may end up with a situation where a latter date such as date of discharge, was masked to be earlier than date of admission. If we were dealing with date of birth and date of death we may end up masking the values in a way that turns an 80 year old into a 5 month old. To this end, the Dependent Date Shift algorithm provides a way to mask these dependent dates in a way that:

- maintains the relationship between the dates (ie: the later date always stays later)
- maintains an approximate interval between the dates, within a provided `intervalRange` / `unit` combination

The Dependent Date Shift algorithm takes in 2 dates (designated `date1` and `date2`). It masks `date1` based on the provided values for `minRange`, `maxRange`, `unit` and `roll`. It then modifies the original interval based on `intervalRange` and unit to calculate `date2`. If the dates differ but the returned interval is zero (i.e.: the difference between the dates is smaller than the interval value), we assume the interval value to be 1 if `date2` is later than `date1` and -1 if `date1` is later than `date2`.

The masked results are deterministic for each pair of inputs with the same algorithm key and date and interval ranges. The algorithm does not allow for zero mask so all masked values will never be equal to the input. If `date1` is not provided, `date2` will be masked based on the provided values for `minRange`, `maxRange`, `unit` and `roll`.

Creating a Dependent Date Shift Algorithm via UI

Select Framework

Secure Lookup	○
Character Mapping	○
Payment Card	○
Date	○
Dependent Date Shift	●
Segment Mapping (legacy)	○
Mapping	○
Binary Lookup	○
Tokenization	○
Min Max	○
Data Cleansing	○
Free Text Redaction	○

Create Dependent Date Shift Algorithm ⊘ Learn More

Algorithm Name

[   ]

Description

[   ]

Minimum Range

[Min Value   ]

Maximum Range

[Max Value   ]

Interval Range

[Interval   ]

☐ **Roll**

Unit

[DAYS   ▾]

[Cancel] [Save]

1. In the upper right-hand region of the **Algorithm** tab under **Settings**, click **Add Algorithm**.

2. Select **DependentDateShift**. The "Create Dependent Date Shift Algorithm" pane appears.

3. Enter an **Algorithm Name**.

> ℹ️ **Info**
>
> This MUST be unique.

4. Enter a **Description**.

5. Enter a **Minimum Range**. This number represents the smallest number of time units that will be added to `date1` when masking. The range is inclusive of this value. Negative values represent units of time in the past and positive values represent units of time in the future. If `date1` is not provided, this is applied to `date2`.

6. Enter a **Maximum Range**. This number represents the largest number of time units that will be added to `date1` when masking. The range is inclusive of this value. Negative values represent units of time in the past and positive values represent units of time in the future. If `date1` is not provided, this is applied to `date2`.

7. Enter an **Interval Range**. A number representing the +/- range value to shift the interval inclusive of the range value. A value of 0 will not change the interval between dates. This number may not be less than 0. If the specified unit difference between date1 and date2 is within the bound of the intervalRange, only values will be provided such that the sign of the difference is preserved. For example, if the day difference between `date1` and `date2` is 2 and the specified intervalRange is 3, only values greater than -2 will be used (i.e.: -1 to 3). Otherwise, the full range of values will be used (i.e.: -3 to 3).

8. Check the **Roll** box to preserve all units of time larger and smaller than the specified unit. Only the value of the specified unit will change. This option is supported for units months, days, hours, minutes, and seconds. This applies when masking `date1`. If `date1` is not provided, this is applied to `date2`.

9. Choose the **Unit** of time from the drop-down: **Years**, **Months**, **Days**, **Hours**, **Minutes**, or **Seconds**. This represents the unit of time the range is expressed in. This unit is also used to determine the interval between `date1` and `date2`.

10. When you are finished, click **Save**.

For information on creating Date Shift algorithms through the API, see API Calls for Creating Algorithms - Dependent Date Shift.

Examples

As an example, a Dependent Date Shift algorithm with a minimum value of 3 and a maximum value of 5 and an interval Range of 5 with the unit set to **Days** will shift the date1 input value by 3 to 5 days into the future. It will then change the interval by a range of +/-5 days from the original interval to mask date2. Dates may mask as follows:

- 1905-12-10 00:00:00, 1907-08-01 10:14:00 → 1905-12-13 00:00:00, 1907-08-06 00:00:00

- 2001-07-31 23:45:30, 2005-04-12 07:13:00 → 2001-08-03 23:45:30, 2005-04-12 23:45:30

- 2021-02-03 12:30:00, 2021-02-07 12:34:00 → 2021-02-06 12:30:00, 2021-02-14 12:30:00

With roll enabled and the same configuration, a date at the end of a month will wrap around to the beginning of the month. Dates may mask as follows:

- 1905-12-10 00:00:00, 1907-08-01 10:14:00 → 1905-12-13 00:00:00, 1907-08-04 00:00:00

- 2001-07-31 23:45:30, 2005-04-12 07:13:00 → 2001-07-03 23:45:30, 2005-03-18 23:45:30

- 2021-02-03 12:30:00, 2021-02-07 12:34:00 → 2021-02-06 12:30:00, 2021-02-14 12:30:00

All inputs with the same value masked with the same algorithm configuration will result in the same output values.

Email

> **Extensible Algorithm Framework**

The Email framework masks string values by splitting the input on the '@' symbol and independently masking the name and domain portions of the email address. Masked values are calculated algorithmically using the algorithm's key, so rekeying the algorithm will cause different outputs to be generated for each input. All inputs to this framework are valid and the framework will not generate non-conformant data events. Note that it is possible for chained algorithms specified for the *Algorithm* option to generate non-conformant data events.

Malformed Input Handling

- Inputs without an '@' symbol: apply the name action to the entire input
- Inputs with no name portion: apply the domain action to the entire input
- Inputs with no domain portion: apply the name action to the entire input
- Inputs with no name portion and no domain portion: return an '@' symbol

Creating an Email Algorithm via UI

Select Framework

Create Email Algorithm ⃝ Learn More

Select Framework		Create Email Algorithm

Select Framework

Secure Lookup ○

Character Mapping ○

Payment Card ○

Date ○

Dependent Date Shift ○

Name ○

Full Name ○

Email ⦿

Segment Mapping (legacy) ○

Mapping ○

Binary Lookup ○

Tokenization ○

Min Max ○

Data Cleansing ○

Free Text Redaction ○

Create Email Algorithm ⃝ Learn More

Algorithm Name

[   ]

Description

[   ]

Mask Name With

[Unique Value   ▼]

Mask Domain With

[Replacement text   ▼]

Domain Replacement

[Enter text like example.com   ]

[Cancel]   [Save]

1. In the upper right-hand region of the **Algorithm** tab under **Settings**, click **Add Algorithm**.

2. Select **Email**. The "Create Email Algorithm" pane appears.

3. Enter an **Algorithm Name**.

> ℹ **Info**
>
> This MUST be unique.

4. Enter a **Description**.

5. From the dropdown **Mask Name With**, choose one of the following options:

- **Unique Value**: applies a SHA-256 hash of the entire input then Base32 encodes the hash value

- **Lookup Value**: applies a secure lookup using the values provided in the uploaded file or file reference

- **Algorithm**: applies the specified string type extensible algorithm

6. If applicable, complete the configuration for masking the name portion as follows:

    - **Lookup Value**: upload a lookup file with new line separated values or provide a file reference

    - **Algorithm**: select a string type extensible algorithm to be used to mask the name portion of the input

7. From the dropdown **Mask Domain With**, choose one of the following options:

    - **Replacement Text**: replaces the domain portion with a fixed value

    - **Algorithm**: applies the specified extensible algorithm instance

8. Complete the configuration for masking the domain portion as follows:

    - **Replacement Text**: enter a value to replace the entire domain portion

    - **Algorithm**: applies the specified extensible algorithm instance

9. When you are finished, click **Save**.

For information on creating Email algorithms through the API, see API Calls for Creating Algorithms - Email.

Examples

As an example, an Email algorithm that uses *Lookup Value* to mask the name portion and *Replacement Text* to mask the domain portion with the following configuration:

Lookup File:

```
Amy
Bob
Jake
Katherine
```

Replacement Text: example.com

May mask as follows:

- "albert@delphix.com" → "Bob@example.com"

- "albert@gmail.com" → "Bob@example.com"

- "andrew_smith_123@delphix.com" → "Katherine@example.com"

Another example that uses the *Algorithm* option for both the name and domain portion with the following configuration:

Name Algorithm: dlpx-core:FirstName

Domain Algorithm: dlpx-core:CM Alpha-Numeric

May mask as follows:

- "bob@gmail.com" → "alton@dqpnx.fsy"

- "bob@hotmail.com" → "alton@poatzdw.bya"

- "alex@gmail.com" → "jameel@dqpnx.fsy"

- "joe_123@yahoo.com" → "miryam@wbpaq.kts"

> **ⓘ Info**
>
> The Email framework will not generate non-conformant data events, but the chained algorithm may generate such events.

All inputs with the same value masked with the same algorithm configuration will result in the same output values.

- "alex@gmail.com" → "jameel@dqpnx.fsy"

- "joe_123@yahoo.com" → "miryam@wbpaq.kts"

Free Text Redaction

A Free Text Redaction algorithm helps you remove sensitive data that appears in free-text columns such as "Notes." This type of algorithm requires some expertise to use because you must set it to recognize sensitive data within a block of text.

One challenge is that individual words might not be sensitive on their own, but together they can be. The algorithm uses profiler sets to determine what information it needs to mask. You can decide which expressions the algorithm uses to search for material such as addresses. For example, you can set the algorithm to look for "St," "Cir," "Blvd," and other words that suggest an address. You can also use pattern matching to identify potentially sensitive information. For example, a number that takes the form 123-45-6789 is likely to be a Social Security Number.

You can use a Free Text Redaction algorithm to show or hide information by displaying either a "denylist" or an "allowlist."

Denylist – Designated material will be redacted (removed). For example, you can set a deny list to hide patient names and addresses. The deny list feature will match the data in the lookup file to the input file.

Allowlist – ONLY designated material will be visible. For example, if a drug company wants to assess how often a particular drug is being prescribed, you can use an allow list so that only the name of the drug will appear in the notes. The allow list feature enables you to mask data using both the lookup file and a profile set.

For either option, a list of words can be imported from an external text file, or alternatively, you can use Profiler Sets to match words based on regular expressions, defined within Profiler Expressions. You can also specify the redaction value that will replace the masked words. Regular expressions defined using Profiler Sets will match individual words within the input text, rather than phrases.

Creating a Free Text Redaction Algorithm via UI

1. Enter an **Algorithm Name**.

2. Enter a **Description**.

3. Select the **Deny List** or **Allow List** radio button.

4. Select **Lookup File** and enter **Redaction Value** OR/AND

5. Select **Profiler Sets** from the drop-down menu and enter **Redaction Value**.

6. Click **Save**.

For information on creating Free Text Redaction algorithms through the API, see API Calls for Creating Algorithms - Free Text Redaction.

Examples

1. Create an Input file.

2. Create an Input file using Notepad. Enter the following text:

> The customer Bob Jones is satisfied with the terms of the sales
> agreement. Please call to confirm at 718-223-7896.

3. Save the file as txt.

4. Create lookup file.

   a. Create a lookup file.

   b. Use Notepad to create a text file and save the file as a txt. Be sure to hit return after each field. The lookup flat file contains the following data:

```
Bob
Jones
Agreement
```

CREATE AN ALGORITHM

You will be prompted for the following information:

1. For **Algorithm Name**, enter **Denylist_Test1**.

2. For **Description**, enter **Denylist Test**.

3. Select the **Deny List** radio button.

4. Select **LookUp File**.

5. Enter redaction value **XXXX**.

6. Click **Save**.

CREATE RULE SET

1. From the job page go to Rule Set and Click **Create Rule Set**.

2. For **Rule Set Name**, enter **Free_ Text_RS**.

3. From the **Connector** drop-down menu, select **Free Text**.

4. Select the **Input File** by clicking the box next to your input file

5. Click **Save**.

CREATE MASKING JOB

1. Use Free_Text Rule Set

2. Execute Masking job.

The results of the masking job will show the following:

```
The customer xxxx xxxx is satisfied with the terms
of the sales xxxx. Please call to confirm at 718-223-7896.
```

"Bob," "Jones," and "agreement" are redacted.

Full Name

| **Extensible Algorithm Framework**

The Full Name algorithm (introduced in Masking Engine version 6.0.8.0) has a logic of recognizing the parts of the input related to the First and Last names, as well as treating the particles (which are imported from the chained Last Name algorithm instance). *Last Name* also has a logic of limiting the number of masked first names (removing the rest), as well as smart trimming of the result (masked) output to the required length.

After distinguishing parts of the input string - Full Name algorithm feeds the single words from the first name part (which also includes middle names, treated same as first names) to the instance of the First Name algorithm and the whole last name part to the instance of the Last Name algorithm. Then it combines the masking results, according the embedded logic and the configuration.
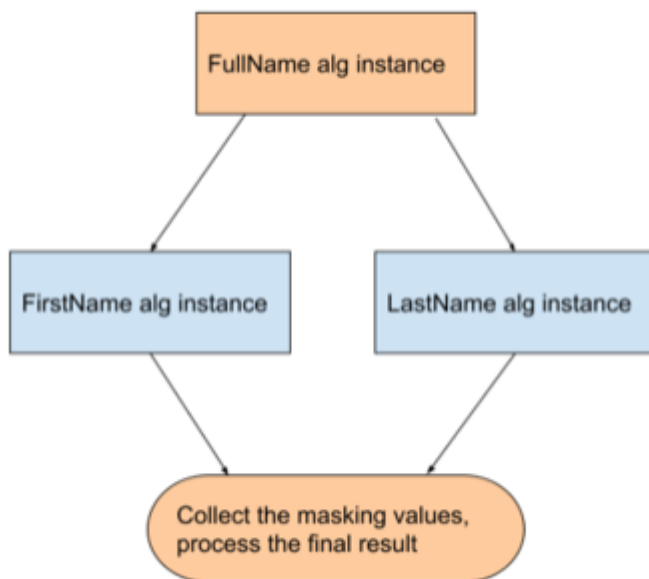
If input string contains only single word - this word is considered as a first name or last name (depending on the **Consider Single Word Input as Last Name** flag) and forwarded for masking to corresponding chained algorithm instance. Single word input is always masked, even if contains configured particle.

Main features of the Full Name Framework:

- Deterministic output: The masked result for each input is consistent when using the same algorithm key, same configuration and same chained algorithm instances.

- Not unique: The masked result might be the same for different inputs.

- Garbage in garbage out: the algorithm returns the unmasked input / null / empty string if input is one of the following: null, empty string "", white spaces only " ", single not alphanumeric symbol (for example "!").

- Single word input: considered either as a Last Name (default) or as a First Name (even if configured in one of the particles files).

- When particle is configured in both particles files: the remove action takes precedence.

- Multiple first names: masks only first N names (1-4, as configured, default = 2), the rest are ignored.

- Full Name Convention: if configured last name separator is detected or configured convention is "last-first-middle" than detects an input as last-first-middle, otherwise first-middle-last (default). Heading/Trailing white spaces are not preserved.

- Smart trim: if trimming of the masked value is required it's done in a way to keep the realistically looking full name as long as possible. For instance: first we trim the heading/trailing preserved particles. If not enough - abbreviating the masked first/middle names (one by one, starting the last one). If still no enough - removing the particles prior to the last name, etc.

Below is an example of smart trim. Let's suppose our masked result (prior to checking of the maxLength) is:

```
"President George Herbert Walker Van Bush Jr."
```

maxLengthOfMaskedFullName value	action	result
55	Nothing. The string is shorter.	President George Herbert Walker Van Bush Jr.
42	Cut particle at the end (if known)	President George Herbert Walker Van Bush
30	Cut particle at the beginning (if known)	George Herbert Walker Van Bush
26	Abbreviate FNs starting last	George Herbert W. Van Bush
22	Continue abbreviate FNs	George H. W. Van Bush
17	Continue abbreviate FNs	G. H. W. Van Bush
14	Cut FN abbreviation(s) starting last	G. H. Van Bush
11	Continue cutting abbreviations	G. Van Bush
8	Leave LN if possible	Van Bush
4	Leave LN if possible	Bush
2	Cut the LN from the end	Bu

Requirement for the chained instances for First Name and Last name masking:
- should be existing extensible algorithm instance, masking the String type.

Although it can be any String type extensible algorithm instance, it is recommended using the instances based on the Name framework

Creating a Full Name Algorithm via UI

1. In the upper right-hand corner of the **Algorithm** tab, click **Add Algorithm**.

2. Choose **Secure Lookup Algorithm**. The Create SL Algorithm pane appears.

3. Enter an **Algorithm Name**. (Required)

> **ⓘ Info**
>
> This MUST be unique on the Masking Engine.

4. Enter a **Description**. (Optional)

5. Choose the **First Name Algorithm**. (Required) In the dropdown menu you will be suggested to choose from the existing extensible algorithms of String type.

6. Choose the **Last Name Algorithm**. (Required) In the dropdown menu you will be suggested to choose from the existing extensible algorithms of String type.

7. Choose the **Maximum First Names** configuration. (Optional. Integer. min value = 1, max value = 4, default = 2) Total number of first/middle names to be masked. The rest would be ignored.

8. Choose the **Maximum Masked Full Name Length** configuration. (Optional. Integer. Default is 0)

   This number should be >= 0 (i.e. not negative). That's the maximum number of characters masked result should fit. I.e. masked result is trimmed (please find above an explanation on smart Full Name trimming) to that length. Value 0 means length is unlimited.s

> **ⓘ Info**
>
> We're also trying to detect the length of the destination field. Some Data Sources provide that value, while others don't. For example: if Data Source provides value **10** for the destination column length and current configuration field is set to **0** or any value longer than 10 - the shortest value wins, i.e. in this example masked result would be trimmed to 10 characters.

9. Specify a default **Full Name Convention**. (Optional. Enum. Default: "First-Middle-Last") Drowpdown menu provides choice of 2 values:

```
First-Middle-Last
Last-First-Middle
```

This configurations helps to the Full Name algorithm to distinguish between first name(s) and last name, if **Last Name Separator(s)** are not configured or not detected in the input string.

10. Choose the **Consider Single Word Input as Last Name**. (Optional. Boolean. Default is true) If chosen (default case) - consider the single word input as a last name. Otherwise as a first name.

11. Configure **Last Name Separators** (Optional. List. Default: contains comma ',') Here you can specify comma separated single punctuation marks (but hyphen '-' and dot '-', which are reserved for another logic) which will serve for identifying the last name in the input. First identified separator makes that distinguishing, rest are ignored. To choose comma ',' there is a separate field aside **Include comma**. By default comma is included as a separator.

    Here is an example of how last name separator works:
    Let's suppose our configured separators are comma ',' and colon ':'.
    Input: "dela Cruz, Maria Cristina: Manansala"
    The first detected separator (framework reads the input left to right) is after word "Cruz".
    So "dela Cruz" will be detected as a last name part, and "Maria Cristina: Manansala" as a first names.
    Masking result would be in the same order with the same separator, for example: "Maritnas, Antonio Stephan".

12. When you are finished, click **Save**.

For the description of any configurable field you can open a popup window by pressing on the blue "? Learn More" link in the upper right corner:

Full Name Algorithm

This is the framework to cover the scenarios where it is required to mask a full name input string with deterministic and not unique masking results. The goal of a Full name masking framework is to mask the first, middle and last names consistently when they appear in the same field using algorithms that can be applied to the component parts (e.g. First Name alone.).

Framework Options:

First Name Algorithm (Required): String type Extensible Algorithm instance to be used to mask first and middle name of the full name input.

Last Name Algorithm (Required): String type Extensible Algorithm instance to be used to mask last name of the full name input.

Maximum First Names: Total number of first/middle names to be masked, the rest would be ignored. Minimum value is 1, maximum is 4 and **default** is 2;

Maximum Masked Full Name Length: Max number of characters or length of masked output string. **default** is 0, which means unlimited.

Full Name Convention: A flag to configure the input full name convention of pattern

- **First-Middle-Last (Default) -** The input name starts with a first name.
- **Last-First-Middle -** The input name starts with a Last name.

Consider Single Word Input as Last Name: A flag to configure if single word input should be masked with Last name or not. **default** is on/true.

Last Name Separators: A comma separated list of characters which should be considered as first and last name separators.

Include comma: A flag to include "," as last name separator character.

OK

For information on creating Full Name algorithms through the API, see API Calls for Creating Algorithms - Full Name.

Mapping

A Mapping algorithm allows you to state what values will replace the original data. It sequentially maps original data values to masked values that are pre-populated to a lookup table through the Masking Engine user interface. There will be no collisions in the masked data because it always matches the same input to the same output. For example "David" will always become "Ragu," and "Melissa" will always become "Jasmine." The algorithm checks whether an input has already been mapped; if so, the algorithm changes the data to its designated output.

You can use a Mapping algorithm on any set of values, of any length, but you must know how many values you plan to mask. You must supply AT MINIMUM the same number of values as the number of unique values you are masking; more is acceptable. For example, if there are 10,000 unique values in the column you are masking you must give the Mapping algorithm AT LEAST 10,000 values.

> **ℹ Info**
>
> When you use a Mapping algorithm, you cannot mask more than one table at a time. You must mask tables serially.

Creating a Mapping Algorithm via UI

1. In the upper right-hand corner of the **Algorithm** tab, click **Add Algorithm**.

2. Select **Mapping Algorithm**.

3. The **Create Mapping Algorithm** pane appears.

4. Enter an **Algorithm Name**.

> **ℹ Info**
>
> This MUST be unique.

5. Enter a **Description**.

6. Specify a **Lookup File**.

7. The value file must have NO header. Make sure there are no spaces or returns at the end of the last line in the file. The following is sample file content. Notice that there is no header and only a list of values.

```
Smallville
Clarkville
Farmville
Townville
Cityname
Citytown
Towneaster
```

8. To ignore specific characters, enter one or more characters in the **Ignore Character List** box. Separate values with a comma.

9. To ignore the comma character (,), select the **Ignore comma (,)** checkbox.

10. When you are finished, click **Save**.

Before you can use the algorithm by specifying it in a profiling job, you must add it to a domain. If you are not using the Masking Engine Profiler to create your inventory, you do not need to associate the algorithm with a domain.

For information on creating Mapping algorithms through the API, see API Calls for Creating Algorithms - Mapping.

Min Max

The Delphix Masking Engine provides a "Min Max algorithm" to normalize data within a range – for example, 10 to 400. Values that are extremely high or low in certain categories allow viewers to infer someone's identity, even if their name has been masked. For example, a salary of $1 suggests a company's CEO, and some age ranges suggest higher insurance risk. You can use a Min Max algorithm to move all values of this kind into the midrange. This algorithm allows you to make sure that all the values in the database are within a specified range.

If the **Out of range Replacement Values** checkbox is selected, a default value is used when the input cannot be evaluated.

Creating a Min Max Algorithm via UI

1. Enter the **Algorithm Name**.

   > **ℹ Info**
   >
   > This MUST be unique.

2. Enter a **Description**.

3. Enter **Min Value** and **Max Value**.

4. Click **Out of range Replacement Values**.

5. Click **Save**.

For information on creating Min Max algorithms through the API, see API Calls for Creating Algorithms - Min Max.

Examples

Example: Age less than 18 years - enter Min Value 0 and Max Value 18.

Name

> **Extensible Algorithm Framework**

Starting in version 6.0.8.0, Delphix has introduced a builtin Extensible *Name* Algorithm Framework, co-existing with the legacy *FIRST NAME SL* and *LAST NAME SL* ones. Name Framework provides masking functionality for String type input. It's based on Secure Lookup mechanism, and includes additional configuration flags making it more flexible and robust.

Similar to Secure Lookup it creates masking results which are determenistic (i.e. the same algorithm with the same configuration and security key will provide the same result for the same input) and not unique. So it you are looking for the framework whose algorithm(s) will provide a unique masking results you should consider using other frameworks (for example Character Mapping).

The new framework uses SHA256 hashing method and allows case configurations for input and output (i.e. masked) values. It also allows filtering accents, configuring the maximum length of the masked value. If input name is a multi-word string it might contain particles, related to the name. By particles we consider any prefixes, suffixes, titles, etc. The new framework allows configuring which particles to be removed, and which to be preserved.

Creating a Name Algorithm via UI

1. In the upper right-hand corner of the **Algorithm** tab, click **Add Algorithm**.

2. Choose **Name** Framework. The **Create Name Algorithm** pane appears.

3. Enter an **Algorithm Name**. (Required)

> **ⓘ Info**
>
> This MUST be unique on the Masking Engine.

4. Enter a **Description**. (Optional)

5. Choose the **Case Sensitive Lookup** configuration. (Optional. Boolean. Default is *false*)

   If **Case Sensitive Lookup** box is marked than the same input of different cases will be masked to the different values. For example:

   ```
   Peter -> John
   peter -> Andrew
   ```

   Otherwise it will be masked to the same values, for example:

```
Peter -> John
peter -> John
```

6. Choose the **Filter Accent** configuration. (Optional. Boolean. Default is *true*)

   If **Filter Accent** box is marked than the similar input with and without accented symbols will be masked to the same values. For example:

```
Adrián -> John
Adrian -> John
```

   Otherwise it will be masked to the different values, for example:

```
Adrián -> John
Adrian -> Peter
```

7. Choose the **Output (Masked) Case** configuration. (Optional. Enum. Default is *Preserve Input Case*)

   It is explained with the examples in the information popup window, which may be opened by clicking on the blue **"? Learn More"** sign on the above Create SL Algorithm window:

Name Algorithm

This is the framework to cover the scenarios where it is required to mask string with deterministic and not unique masking results

Framework Options:

Output (Masked) Case:

- **Preserve Lookup File Case -** keep masked value as found in Lookup File
- **Preserve Input Case (Default)-** check the input case, which can be one of following three:
  - All uppercase - in that case force whole masked value to uppercase
  - All lowercase - in that case force whole masked value to lowercase
  - Mixed (if at least 1 character case is different from others) - in that case keep masked value as found in Lookup File
- **Force all Uppercase -** forces whole masked value to uppercase
- **Force all Lowercase -** forces whole masked value to lowercase

Maximum Masked Name Length: Max number of characters or length of masked output string. **default** is 0, which means unlimited.

Case Sensitive Lookup: A flag to configure if input value case should be considered for Lookup match or not. **default** is off/false.

Filter Accent: A flag to configure if accent character in input should be considered for Lookup match or not. **default** is on/true.

Particles to Preserve (Optional): A file with list of words those should be preserved while masking. ex; "Mr.","Mrs.","Sir" etc.

Particles to Remove (Optional): A file with list of words those should be removed while masking. ex; "Mr.","Mrs.","Sir" etc.

Lookup File (Required): A file with list of values for masking.

OK

1. Choose the **Maximum Masked Name Length** configuration. (Optional. Integer. Default is 0)

   This number should be >= 0 (i.e. not negative). That's the maximum number of characters masked result should fit. I.e. masked result is trimmed to that length. Value 0 means length is unlimited.

> **ℹ Info**
>
> We're also trying to detect the length of the destination field. Some Data Sources provide that value, while others don't. For example: if Data Source provides value **10** for the destination column length and current configuration field is set to **0** or any value longer than 10 - the shortest value wins, i.e. in this example masked result would be trimmed to 10 characters.

> **⚠ Warning**
>
> Some UTF-8 characters might take 2 bytes. If lookup file contains those characters - the trimmed result might be not as expected, since we trim by the number of characters and not number of bytes. There is a bug open for that mismatch.

2. Specify a **Particles to Preserve** File. (Optional. Locally chosen file, or a FileReference) Contains particles to be preserved. I.e. those particles are not masked. For example if file contains particle "von" and "Froum" is masked to "Smith" than

```
von Froum -> von Smith
```

3. Specify a **Particles to Remove** File. (Optional. Locally chosen file, or a FileReference) Contains particles to be removed. Those particles are removed prior to masking, i.e. they do not affect masking result. For example if file contains particle "von" and "Froum" is masked to "Smith" than

```
von Froum -> Smith
Froum -> Smith
```

> **ℹ Info**
>
> If particle is found in both "Preserve" and "Remove" files - it will be removed.

4. Specify a **Lookup File**. (Required. Locally chosen file, or a FileReference)

   This file is a single list of values. It does not require a header. Every line of the Lookup File might be used as a masked value. The Lookup File must be ASCII or UTF-8 encoding compatible. The following is sample file content:

```
Ann
Marie
Tomas
Ann-Marie
Basil
Mark
```

5. When you are finished, click **Save**.

For information on creating Name algorithms through the API, see API Calls for Creating Algorithms - Name.

Payment Card

Extensible Algorithm Framework

The Payment Card framework masks payment card numbers based on the starting digits to be preserved and the minimum number of positions to be masked. This framework is built on top of the Character Mapping Algorithm Framework with a character set of [0-9]. All characters outside of this character group remain unmasked. Masked values are calculated algorithmically using the algorithm's key, so rekeying the algorithm will cause different outputs to be generated for each input. The last digit may remain the same if the calculated check digit is equivalent to the last digit of the input. Any inputs with more than one digit will never mask to the original value.

> ⚠️ **Warning**
>
> Any inputs with a single digit will remain unmasked.

This framework preserves the validity of the payment card number using the Luhn check. All input values with valid Luhn checks will be masked to values with valid Luhn checks. All invalid values with invalid Luhn checks will be masked to values with invalid Luhn checks.

Creating a Payment Algorithm via UI

Select Framework

Secure Lookup	○
Character Mapping	○
Payment Card	●
Date	○
Dependent Date Shift	○
Segment Mapping (legacy)	○
Mapping	○
Binary Lookup	○
Tokenization	○
Min Max	○
Data Cleansing	○
Free Text Redaction	○

Create Payment Card Algorithm

Algorithm Name

Description

Minimum Masked Positions

1

Preserve Starting Digits

0

Cancel   Save

1. In the upper right-hand region of the **Algorithm** tab under **Settings**, click **Add Algorithm**.

2. Select **Payment Card**. The "Create Payment Card Algorithm" pane appears.

3. Enter an **Algorithm Name**.

> ℹ **Info**
>
> This MUST be unique.

4. Enter a **Description**.

5. Set **Minimum Masked Positions**. This value is the minimum number of positions that must be replaced for masking to be considered successful. If fewer positions are masked, a non- conforming data handling error is triggered. Values for this field must be in the range [0-32].

6. Set **Preserve Starting Digits**. This value specifies how many maskable characters should be preserved from the beginning of the input. Only maskable characters are included in this count. Values for this field must be in the range [0-32].

7. When you are finished, click **Save**.

For information on creating Payment Card algorithms through the API, see API Calls for Creating Algorithms - Payment Card.

Examples

As an example, a Payment Card algorithm with a *minMaskedPositions* value of 6 and a *preserve* value of 6 may mask as follows:

- "5419033646326699" → "5419036803270758"

- "5419-0336-4632-6699" → "5419-0368-0327-0758"

- "5319abc0339def4632ghi6599!" → "5319abc0364def1507ghi4137!"

All inputs with the same sequence of digits masked with the same algorithm configuration will result in the same output values.

Regex Decompose

> **Extensible Algorithm Framework**

The Regex Decompose framework masks values that match specified Java 8 regular expressions. The algorithm attempts to match the algorithm input against each regular expression, and once a match is found, the associated action is applied to transform either the entire input, or each capturing group (parts of the input) defined by the expression. A fallback action may be provided for use when none of the defined regular expressions match the input. If no fallback action is defined and an input fails to match any of the defined regular expressions, the algorithm may be configured to generate a non-conformant data exception.

Capturing groups are used in regular expressions to create subgroups. These can be expressed in regular expressions using parentheses to group characters together. This algorithm allows for different capturing groups to be assigned different mask actions. Nested capturing groups are unsupported and may lead to unpredictable behavior. If no capturing groups are defined, the first action is applied to the entire match. In this case, the action list should contain only one action.

Creation of Regex Decompose algorithms can only be done through the API, see API Calls for Creating Algorithms - Regex Decompose.

Examples

As an example, a Regex Decompose algorithm with the following configuration:

```
Mask Pattern:
    Regular Expression: "[0-9]*"
    Action: Redact
    Redact String: "redacted"
Require Mask: false
Trim Input: true
Maximum Input Length: 10
```

Will produced masked results as follows:

- "12345" → "redacted"

- " 6789 " → " redacted "

- "12345678901" → non-conformant data

  - exceeds maximum input length

- "abc123" → "abc123"

  - remains unmasked since it does not match the regex pattern

The provided regular expression matches any inputs with 0 or more digits in the range [0-9] and any inputs that match will be replaced with the string "redacted". Any inputs that contain characters outside of the range [0-9] will not be masked. If require mask was set to true, the last example "abc123" would trigger a non-conformant data event as the value would not be masked by the algorithm.

Another example that includes capturing groups with the following configuration:

```
Mask Pattern:
    Regular Expression: "([1-9]*)-([a-z]*)"
    Action 1: Redact
        Redact Character: 'X'
    Action 2: Preserve
Require Mask: true
Trim Input: true
Maximum Input Length: 10
Fallback Action: Redact
    Redact String: "redacted"
```

Will produce masked results as follows:

- "12345-abc" → "XXXXX-abc"

- "abc-123" → "redacted"

  - does not match the pattern so the fallback action is applied

- "1-a" → "X-a"

- "-" → "redacted"

  - does match the pattern but the masked output would be "-" which breaks the requirement that the output must be different from the input so the fallback action is applied

- "redacted" → non-conformant data

  - does not match the pattern so the fallback action is applied but the fallback action does not change the value so it fails the requirement that the input must be masked

The provided regular expression matches any inputs with 0 or more digits in the range [1-9], a dash, and 0 or more characters in the range [a-z]. Any inputs that do not match that pattern will be masked by the fallback action. If the fallback action fails to change the input, a non-conformant data event will occur.

All inputs with the same input value masked with the same algorithm configuration will result in the same output values.

Secure Lookup

| **Extensible Algorithm Framework**

Secure Lookup is the most commonly used type of algorithm. It is easy to generate and works with different languages. When this algorithm replaces real, sensitive data with fictional data, it is possible that it will create repeating data patterns, known as "collisions." For example, the names "Tom" and "Peter" could both be masked as "Matt". Because names and addresses naturally recur in real data, this mimics an actual data set. However, if you want the Masking Engine to mask all data into unique outputs, you should use Character Mapping.

Starting in version 6.0.4.0, we introduced a builtin Extensible Secure Lookup Algorithm Framework, co-existing with the legacy one. The new framework uses SHA256 hashing method and allows case configurations for input and output (i.e. masked) values.

Creating a Secure Lookup Algorithm via UI

1. In the upper right-hand corner of the **Algorithm** tab, click **Add Algorithm**.

2. Choose **Secure Lookup Algorithm**. The Create SL Algorithm pane appears.

3. Choose the type of SL algorithm framework to use. For Legacy one click on `Legacy` button at the top. The default version is the newer Extensible SL Framework, allowing configuration of the following options:

```
Output (Masked) Case
Case Sensitive Lookup
```

which are disabled for Legacy SL Framework type.

4. Enter an **Algorithm Name**.

> ℹ️ **Info**
>
> This MUST be unique.

5. Enter a **Description**.

6. Choose the **Output (Masked) Case** configuration. This option is available only for the Extensible SL Framework type. It is explained with the examples in the information popup window, which may be opened by clicking on the blue question sign on the above Create SL Algorithm window:

SL Algorithm Information

With the new SL, the new algorithm will be able to customize given properties, if you want to continue with the legacy behavior please check the legacy option and properties will be set to default old behavior.

Output (Masked) Case options are as below:

- **Preserve Lookup File Case** - keep masked value as found in Lookup File
- **Preserve Input Case** - check the input case, which can be one of following three:
    - All uppercase - in that case force whole masked value to uppercase
    - All lowercase - in that case force whole masked value to lowercase
    - Mixed (if at least 1 character case is different from others) - in that case keep masked value as found in Lookup File
- **Force all Uppercase** - forces whole masked value to uppercase
- **Force all Lowercase** - forces whole masked value to lowercase

Example:

Output value case option value	Input value	LookupFile value	Resulting masked value
Preserve Lookup File	Bernard	Michael	Michael
	Bernard	Michael	Michael
	beRnard	Michael	Michael
Preserve Input	Bernard	Michael	Michael
	bernard	Michael	michael
	BERNARD	Michael	MICHAEL
	beRnard	Michael	Michael
	samuel	ANTHONY	anthony
	Adam	WilLiam	WilLiam
Force all uppercase	Bernard	Michael	MICHAEL
Force all lowercase	Bernard	Michael	michael

OK

7. Choose the **Case Sensitive Lookup** configuration. This option is available only for the Extensible SL Framework type.

   If **Case Sensitive Lookup** box is marked then the same input of different cases will be masked to the different values. For example:

   ```
   Peter -> John
   peter -> Andrew
   ```

   If that setting is not marked (which is a default option), then lookup would be case insensitive, for example:

   ```
   Peter -> John
   peter -> John
   ```

8. Specify a **Lookup File**.

   This file is a single list of values. It does not require a header. Every line of the Lookup File might be used as a masked value. The Lookup File must be ASCII or UTF-8 encoding compatible. The following is sample file content:

   ```
   Smallville
   Clarkville
   Farmville
   Townville
   Cityname
   Citytown
   Towneaster
   ```

9. When you are finished, click **Save**.

10. Before you can use the algorithm in a profiling job, you must add it to a domain.

ⓘ **Info**

For Legacy Framework type only: If the lookup file contains foreign alphabet characters, the file must be saved in UTF-8 format with no BOM (Byte Order Marker) for the Masking Engine to read the Unicode text correctly. Some applications, e.g. Notepad on Windows, write a BOM (Byte Order Marker) at the beginning of the Unicode file. This character will be included as part of the first replacement value, potentially leading to SQL update or insert errors on databases where this character is not allowed in VARCHAR fields - when trying to run a masking job that applies a Secure Lookup algorithm that has been created based on a UTF-8 file that included a BOM.

Extensible builtin SL Framework filters the BOM automatically without causing the mentioned errors.

For information on creating Secure Lookup algorithms through the API, see API Calls for Creating Algorithms - Secure Lookup.

Segment Mapping

Segment Mapping algorithms produce no overlaps or repetitions in the masked data. They let you create unique masked values by dividing a target value into separate segments and masking each segment individually.

You can mask up to a maximum of 36 values using segment mapping. You might use this method if you need columns with unique values, such as Social Security Numbers, primary key columns, or foreign key columns. When using segment mapping algorithms for primary and foreign keys, in order to make sure they match, you must use the same Segment Mapping algorithm for each. You can set the algorithm to produce alphanumeric results (letters and numbers) or only numbers.

With Segment Mapping, you can set the algorithm to ignore specific characters. For example, you can choose to ignore dashes [-] so that the same Social Security Number will be identified no matter how it is formatted. You can also preserve certain values. For example, to increase the randomness of masked values, you can preserve a single number such as 5 wherever it occurs. Or if you want to leave some information unmasked, such as the last four digits of Social Security numbers, you can preserve that information.

To decide whether Character Mapping or Segment Mapping is the correct option for your use case, see Choosing Between Character and Segment Mapping Frameworks.

Creating a Segment Mapping Algorithm via UI

Select Algorithm

Secure Lookup Algorithm   ○

Segment Mapping Algorithm   ⦿

Mapping Algorithm   ○

Binary Lookup Algorithm   ○

Tokenization Algorithm   ○

Min Max Algorithm   ○

Data Cleansing Algorithm   ○

Free Text Redaction Algorithm ○

Create Segment Mapping Algorithm

Algorithm Name

Description

Number of Segments

2

Segment 1

Numeric   2

Real Values

Min #   Max #   Range #

Mask Values

Min #   Max #   Range #

Segment 2

Numeric   2

Real Values

Min #   Max #   Range #

Mask Values

Min #   Max #   Range #

Ignore Characters Separated by comma(,)

Ignore comma(,) ☐   **Add Control Characters**

Preserve Original Values

Starting Position   Length

                                              Add

If Nonconforming Data is encountered

Use global setting   ▾   ❷ Learn More

Cancel   Save

1. In the upper right-hand region of the **Algorithm** tab, click **Add Algorithm**.

2. Select **Segment Mapping Algorithm**. The Create Segment Mapping Algorithm pane appears.

3. Enter an **Algorithm Name**.

> **ℹ️ Info**
>
> This MUST be unique.

4. Enter a **Description.**

5. From the **No. of Segment** drop-down menu, select how many segments you want to mask.

> **✏️ NOTE**
>
> This number does NOT include the values you want to preserve.

   The minimum number of segments is 2; the maximum is 9. A box appears for each segment.

6. For each segment, choose the **Type** of segment from the drop-down: **Numeric** or **Alphanumeric**.

> **ℹ️ Info**
>
> **Numeric** segments are masked as whole segments. **Alphanumeric** segments are masked by individual characters.

7. For each segment, select its **Length** (number of characters) from the drop-down menu. The maximum is 4.

8. Optionally, for each segment, specify range values. You might need to specify range values to satisfy particular application requirements, for example. See the details below.

9. **Preserve Original Values** by entering **Starting position** and **length** values. (Position starts at 1.) For example, to preserve the second, third, and fourth values, enter Starting position **2** and length **3**.

   If you need additional value fields, click **Add**.

10. To override the behavior of the segment mapping algorithm when it encounters data values in an unexpected format, you can change the selection under **Nonconforming Data behavior**. By default, the segment mapping algorithm will **Use global setting** as specified on the **Algorithm Settings** page. Selecting **Mark job as Failed** will instruct the segment mapping algorithm to throw an exception that will result in the job failing. Selecting **Mark job as Succeeded** will instruct the segment mapping algorithm to ignore the non-conformant data and not throw an exception. Note that **Mark job as Succeeded** will result in the non-conformant data not being masked should the job succeed, but the **Monitor** page will display a warning that can be used to report the non-conformant data events.

11. When you are finished, click **Save**.

12. Before you can use the algorithm in a profiling job, you must add it to a domain. If you are not using the Masking Engine Profiler to create your inventory, you do not need to associate the algorithm with a domain.

Specifying Range Values

You can specify ranges for **Real Values** and **Mask Values**. With Real Values ranges, you can specify all the possible real values to map to the ranges of masked values. Any values NOT listed in the Real Values ranges would then mask to themselves.

Specifying range values is optional. If you need unique values (for example, masking a unique key column), you MUST leave the range values blank. If you plan to certify your data, you must specify range values.

When determining a numeric or alphanumeric range, remember that a narrow range will likely generate duplicate values, which will cause your job to fail.

1. To ignore specific characters, enter one or more characters in the **Ignore Character List** box. Separate values with a comma.

2. To ignore the comma character (,), select the **Ignore comma (,)** checkbox.

3. To ignore control characters, select **Add Control Characters**. The **Add Control Characters** window appears.

4. Select the individual control characters that you would like to ignore, or choose **Select All** or **Select None**.

5. When you are finished, click **Save**.

6. You are returned to the Segment Mapping pane.

Numeric Segment Type

- **Min#** — A number; the first value in the range. Value can be 1 digit or up to the length of the segment. For example, for a 3-digit segment, you can specify 1, 2, or 3 digits. Acceptable characters: 0-9.

- **Max#** — A number; the last value in the range. The value should be the same length as the segment. For example, for a 3-digit segment, you should specify 3-digits. Acceptable characters: 0-9.

- **Range#** — A range of numbers; separate values in this field with a comma (,). Value should be the same length as the segment. For example, for a 3-digit segment, you should specify 3 digits. Acceptable characters: 0-9.

> ℹ️ **Info**
>
> If you do not specify a range, the Masking Engine uses the full range. For example, for a 4-digit segment, the Masking Engine uses 0-9999.

Alphanumeric Segment Type

- **Min#** — A number from 0 to 9; the first value in the range.

- **Max#** — A number from 0 to 9; the last value in the range.

- **MinChar** — A letter from A to Z; the first value in the range.

- **MaxChar** — A letter from A to Z; the last value in the range.

- **Range#** — A range of alphanumeric characters; separate values in this field with a comma (,). Individual values can be a number from 0 to 9 or an uppercase letter from A to Z. (For example, B,C,J,K,Y,Z)

> ℹ️ **Info**
>
> If you do not specify a range, the Masking Engine uses the full range (A-Z, 0-9). If you do not know the format of the input, leave the range fields empty. If you know the format of the input (for example, always alphanumeric followed by numeric), you can enter range values such as A2 and S9.

> ⚠️ **Warning**
>
> The Segment Mapping pattern and sub-patterns need to match the data in order for it to be masked. If the data is longer than the defined pattern it will be passed through unmasked. To avoid this unwanted behavior - patterns (segments), Ignore Characters, and Preserve Original Values should be set to match the data.

For information on creating Segment Mapping algorithms through the API, see API Calls for Creating Algorithms - Segment Mapping.

Examples

Perhaps you have an account number for which you need to create a segment mapping algorithm. You can separate the account number into segments, preserving the first two-character segment, replacing a segment with a specific value, and preserving a hyphen. The following is a sample value for this account number:

NM831026-04

Where:
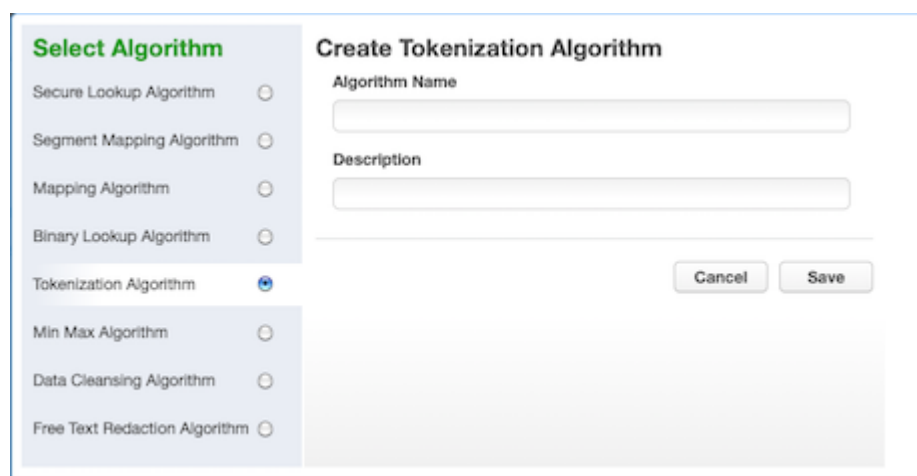
- **NM** is a plan code number that you want to preserve, always a two-character alphanumeric code.

- **831026** is the uniquely identifiable account number. To ensure that you do not inadvertently create actual account numbers, you can replace the first two digits with a sequence that never appears in your account numbers in that location. (For example, you can replace the first two digits with 98 because 98 is never used as the first two digits of an account number.) To do that, you want to split these six digits into two segments.

- **-04** is a location code. You want to preserve the hyphen and you can replace the two digits with a number within a range (in this case, a range of 1 to 77).

Tokenization

A Tokenization algorithm is the only type of algorithm that allows you to reverse its masking. For example, you can use a Tokenization algorithm to mask data before you send it to an external vendor for analysis. The vendor can then identify accounts that need attention without having any access to the original, sensitive data. Once you have the vendor's feedback, you can reverse the masking and take action on the appropriate accounts.

Like Mapping, a Tokenization algorithm creates a unique token for each input such as "David" or "Melissa." The actual data (for example, names and addresses) are converted into tokens that no longer convey any meaning.

Creating a Tokenization Algorithm via UI

1. At the top right of the **Algorithm** tab, click **Add Algorithm**.

2. Select **Tokenization Algorithm**. The Create Tokenization Algorithm pane appears.

3. Enter an **Algorithm Name**.

> ℹ️ **Info**
>
> This MUST be unique.

4. Enter a **Description**.

5. Click **Save**.

Once you have created an algorithm, you will need to associate it with a domain.

1. Navigate to the **Home>Settings>Domains** page and click **Add Domain**.

2. Enter a domain name.

3. From the **Tokenization Algorithm Name** drop-down menu, select your algorithm.

Next, create a Tokenization Environment:

1. On the home page, click **Environments**.

2. Click **Add Environment**.

3. For **Purpose**, select **Tokenize/Re-Identify**.

4. Click **Save**.

> ℹ️ **Info**
>
> This environment will be used to re-identify your data when required.

5. Set up a Tokenization job using the tokenization method. Execute the job.

Create Tokenization Job

Job Name

QA Tokenize

Tokenization Method

Tokenization Method ▾

Target: token test

☐ **Multi Tenant**

Rule Set

token test ▾

Generator

DMsuite ▾

No. of Streams

20

Remote Server

Remote Server ▾

Min Memory   **Max Memory**

In MB   In MB

Update Threads

4

Commit Size   **Feedback Size**

☐ Truncate   ☐ Disable Trigger
☐ Batch Update   ☐ Disable Constraint
☐ Drop Indexes

Prescript

Select...

Postscript

Select...

Comments

Email

Cancel   Save

Examples

Here is example data before and after Tokenization is applied to the address field.

Before Tokenization

```
1 ID,fname,address,ssn
2 1,Erasmus,245 Park Ave,123-45-6789
3 2,Ridley,1003 Stant Drive,123-45-6789
4 3,Jason,45 Omega Suites,123-45-6789
5 4,Waldeve,1 Pulitzer way,123-45-6789
6 5,Salathiel,245 park Ave,123-45-6789
```

After Tokenization

```
1 ID,fname,address,ssn
2 1,Erasmus,L1kgrFFRzafOTUqfpZAmiC==,123-45-6789
3 2,Ridley,+7A16uqP1BSbaaL1f0T7lzqijNVHU38Z2fMMK0fX4+O=,123-45-6789
4 3,Jason,C4v5jrlmKEhKC3acnQKqEk==,123-45-6789
5 4,Waldeve,v89pB9b9QISxyYvs/agYUg==,123-45-6789
6 5,Salathiel,yrLNBhI8j40ld7y7dXRqwY==,123-45-6789
```

Creating Masking Jobs

This section describes how users can create a masking job.

Creating New Jobs

In the **Environment Overview** screen, select one of the jobs icons to create the corresponding job:

- Profile
- Mask
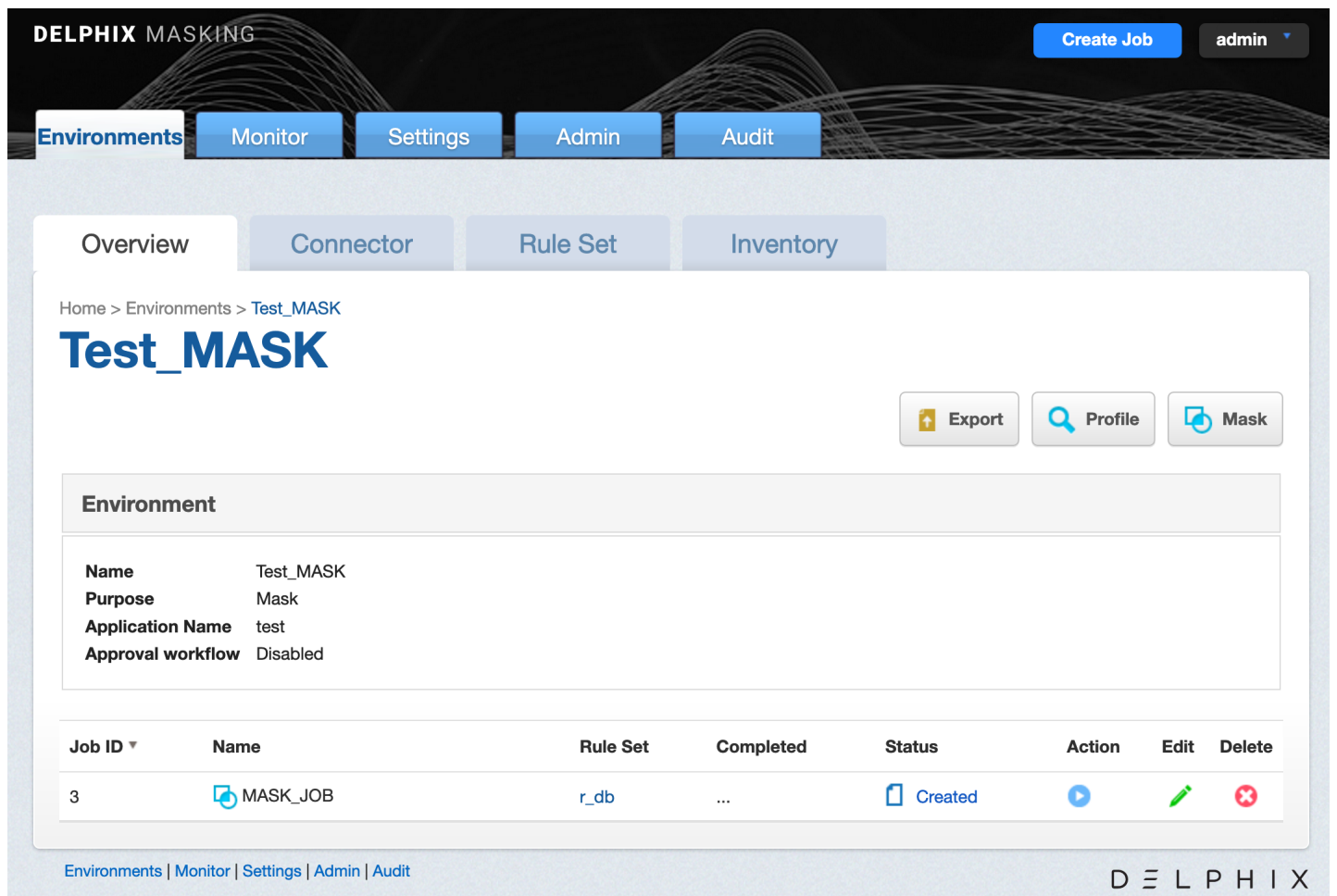
Creating a New Masking Job

To create a new masking job:

1. Click **Mask**. The **Create Masking Job** window appears.

Create Masking Job

Job Name

[   ]

Masking Method

[Masking Method   ▼]

Target: DatabaseTarget

☐ **Multi Tenant**

Rule Set

[Rule Set   ▼]

Streams:
Number   **Row Limit**

[1   ]   [   ]

Min Memory   **Max Memory**

[In MB   ]   [In MB   ]

Update Threads

[1   ]

If Nonconforming Data is encountered

☐ Stop job on first occurrence

Commit Size   **Feedback Size**

[   ]   [   ]

☐ Disable Trigger

☑ Batch Update   ☐ Disable Constraint

☐ Drop Indexes

Prescript

[Select...]

Postscript

[Select...]

Comments

[   ]

Email

[noreply@delphix.com   ]

[Cancel] [Save]

2. You will be prompted for the following information:

   a. **Job Name** — A free-form name for the job you are creating. Must be unique across the entire application.

   b. **Masking Method** — Select either **In-Place** or **On-The-Fly**. **In-Place** jobs update the source environment with the masked values. **On-The-Fly** jobs read unmasked data from the source environment and writes the masked data to the target environment.

> **ⓘ INFO: On-The-Fly Masking Jobs.**
>
> Only certain combinations of connector types are supported. On-The-Fly jobs where the source and target connectors are of the same type (e.g. Oracle to Oracle, delimited file to delimited file), and jobs with a database source (e.g. Oracle, MS SQL) and the target is delimited files are supported.
>
> The target tables or files must be created in advance and the names must match the names of the source tables or files. In the case of a database to delimited file job, the file names should match the table names.

c. **Multi Tenant** — Checkbox if the job is for a multi-tenant database.

> **ⓘ INFO: Provisioning Masked VDBs.**
>
> A job must be Multi-Tenant to use it when creating a masked virtual database (VDB). This option allows existing rulesets to be reused to mask identical schemas via different connectors. The connector can be selected at job execution time.

d. **Rule Set** — Select a rule set that this job will execute against.

e. **Source Environment** (only for On-The-Fly Masking Method) - Select the Source Environment that this job will get the data from.

f. **Source Connector** (only for On-The-Fly Masking Method) - Select the Source Connector that provides the connection to the the chosen Source Environment.

g. **Streams: Number**—The number of parallel streams to use when running the job. For example, you can select two streams to mask two tables in the Rule Set concurrently in the job instead of one table at a time.

> **ⓘ INFO: Choosing the Number of Streams**
>
> Jobs - even with a single stream - will have separate execution threads for input, masking, and output logic. While it is not necessary to increase the number of streams to engage multiple CPU cores in a job, doing so may increase overall job performance dramatically, depending on a number of factors. These factors include the performance characteristics of the data source and target, the number of processor cores available to the Delphix Masking Engine, and the number and types of masking algorithms applied in the Rule Set. The memory requirements for a job increase proportionately with the number of streams.

h. **Streams: Row Limit**—The number of data rows that may be in process simultaneously for each masking stream. For file jobs, this controls the number of delimited or fixed-width lines, mainframe records, or XML elements in process at one time. Setting this value to 0 allows unlimited rows into each stream, while leaving it blank will select a default limit based on job type.

> **ℹ INFO: Choosing the Row Limit**
>
> The default Row Limit values have been selected to allow typical jobs to run successfully with the default job memory and streams number settings. This assumes a maximum row or record size of approximately 2000 bytes with 100 masked columns. If masked row or record size, or column count, exceed these values, it may be necessary to either allocate more memory to the job by increasing Max Memory, or reduce the Row Limit to a smaller value. Conversely, if the masked rows are quite small and have few masking assignments, increasing the Row Limit may improve job performance. Remember to consider the worst case (the largest rows, the most masking assignments) table or file format in the Rule Set when making this determination.

i. **Min Memory (MB)** — Minimum amount of memory to allocate for the job, in megabytes.

j. **Max Memory (MB)** — Maximum amount of memory to allocate for the job, in megabytes.

> **ℹ Info**
>
> It is recommended that the **Min/Max Memory** should be set to at least to **1024**.

k. **Update Threads** — The number of update threads to run in parallel to update the target database.

> **⚠ Warning**
>
> Multiple threads should not be used if the masking job contains any table without an index. Multi-threaded masking jobs can lead to deadlocks on the database engine. Multiple threads can cause database engine deadlocks for databases using T-SQL If masking jobs fail and a deadlock error exists on the database engine, then reduce the number of threads.

l. **Nonconforming Data behavior**

- **Stop job on first occurrence** - (optional) To abort a job on the first occurrence of non-conformant data. The default is for this checkbox to be clear.

> **ℹ Info**
>
> The job behavior depends on the settings specified in the **Algorithm Settings** page and on the individual algorithm pages that define how you view the presence of Nonconforming data. The setting on the **Algorithm Settings** page is global that can be overridden by the setting on the algorithm page for that algorithm. These settings declare if the presence of Nonconforming data is a failure, or a success for the job. If **Mark job as Failed** is selected as a result of the above settings then the job would be aborted on the first occurrence of nonconforming data. If **Mark job as Succeeded** is selected as a result of the above settings then the job will not be aborted.

m. **Commit Size** — (optional) The number of rows to process before issuing a commit to the database.

n. **Feedback Size** — (optional) The number of rows to process before writing a message to the logs. Set this parameter to the appropriate level of detail required for monitoring your job. For example, if you set this number significantly higher than the actual number of rows in a job, the progress for that job will only show 0 or 100%.

o. **Disable Constraint** — (optional) Whether to automatically disable database constraints. The default is for this check box to be clear and therefore not perform automatic disabling of constraints. For more information about database constraints see Enabling and Disabling Database Constraints.

p. **Batch Update** — (optional) Enable or disable use of a batch for updates. A job's statements can either be executed individually, or can be put in a batch file and executed at once, which is faster.

q. **Disable Trigger** — (optional) Whether to automatically disable database triggers. The default is for this check box to be clear and therefore not perform automatic disabling of triggers.

r. **Drop Indexes** — (optional) Whether to automatically drop indexes on columns which are being masked and automatically re-create the index when the masking job is completed. The default is for this check box to be clear and therefore not perform automatic dropping of indexes.

s. **Prescript** — (optional) Specify the full pathname of a file that contains SQL statements to be run before the job starts, or click **Browse** to specify a file. If you are editing the job and a prescript file is already specified, you can click the **Delete** button to remove the file. (The Delete button only appears if a prescript file was already specified.) For information about creating your own prescript files.

t. **Postscript** — (optional) Specify the full pathname of a file that contains SQL statements to be run after the job finishes, or click **Browse** to specify a file. If you are editing the job and a postscript file is already specified, you can click the **Delete** button to remove the file. (The Delete button only appears if a postscript file was already specified.) For information about creating your own postscript files see Creating SQL Statements to Run Before and After Jobs

u. **Comments** — (optional) Add comments related to this masking job.

v. **Email** — (optional) Add e-mail address(es) to which to send status messages.

3. When you are finished, click **Save**.

Enabling and Disabling Database Constraints

Depending on the type of target database you are using, the Delphix Engine can automatically enable and disable database constraints.

The ability to enable and disable constraints ensures that the Delphix Engine can update columns that have primary key or foreign key relationships. You can set Delphix to handle constraints automatically by enabling the **Disable Constraint** checkbox on a Masking job.

> ✏️ **Note**
>
> Delphix does not support the enable/disable constraints feature for all databases. To see which databases are supported, see the Data Source Support page.

Creating SQL Statements to Run Before and After Jobs

When you create a masking job or a certification job, you can specify standard, static SQL statements to run before (prescript) you run a job and/or after (postscript) the job has completed. For example, if you want to mask a column that has a foreign key constraint to another table, you could use a prescript to disable the constraint and a postscript to re-enable the constraint.

You create prescripts and postscripts by creating a text document with the SQL statement(s) to execute. If the text file contains more than one SQL statement, each statement must be separated by a semicolon [;]. For example to remove records with date_column before December 12th, 2017 before masking a table (owner.table), one would create a prescript file containing the following and associate the prescript file to the masking job that includes the table in its ruleset:

```
DELETE FROM owner.table WHERE date_column < '20171207';
```

Database-specific, SQL programming extensions (such as PL/SQL and Transact-SQL) and dynamic SQL statements are not supported in prescripts and postscripts. However, you can create procedures and functions using your database tooling of choice and call them using standard SQL statements from a prescript or postscript.

Managing Jobs from the Environment Overview Screen

Submitting a Job

To submit or resubmit a job from the Environment Overview screen, click the Play icon in the Action column for the desired job.

Upon submitting the job, the masking engine will check if there are enough resources allocated to simultaneously running jobs to determine whether to run or queue the submitted job. There are two resources that the submitted job will be verified against.

1. Maximum memory for all running jobs.

   - This limit defaults to a dynamic calculation of 75% of the entire system's available memory minus 6GB, which is reserved for the masking web application. This calculation can be manually overridden by setting the general application setting `MaximumMemoryForJobs` . To revert a manually overridden limit back to the dynamically calculated limit, set the `MaximumMemoryForJobs` to 0.

2. Maximum number of simultaneously running jobs.

   - This limit defaults to 7 simultaneously running jobs. However, this default value can be overridden by setting the general application setting `NumSimulJobsAllowed` to a different value. The engine also provides a dynamic limit for this resource, which takes the number of available cores on the system minus 1, reserved for the masking engine. This dynamic limit can be used by setting `NumSimulJobsAllowed` to 0.

> ✏️ **Note**
>
> If the submitted job causes all of the currently running jobs to exceed either of those limits, the job will be queued and run at a later time when enough of the other jobs stop running to free up resources. To view the the position of the job in the queue, navigate to the Monitor Screen.

Stopping a Job

The Play icon changes to a Stop icon while the job is RUNNING OR QUEUED.

To stop a RUNNING or QUEUED job from the Environment Overview screen:

1. Locate the job you want to stop.

2. In the job's **Action** column, click the **Stop** icon.

3. A popup appears asking, "Are you sure you want to stop job?" Click **OK**.

4. When the job has been stopped, its status changes to CANCELLED.

Stopping a RUNNING job can result in corrupted or semi-masked data. Stopping a QUEUED job will have no impact on the data source, since the execution of the job has not yet begun. If email notifications are enabled, stopping a QUEUED job will send an email to the user who created the job indicating that it has been cancelled by the user who stopped the job.

Verifying a Job

When the job is complete, the status will change to either SUCCEEDED or FAILED.

After the job completes successfully, return to the Inventory and check that the Domain and Method populated automatically for sensitive data. Sample screenshot below.

Monitoring Masking Job

This section describes how users can monitor the progress of a masking job.

Monitoring masking job refers to the job status or completion state. To determine if the masking operation is completed, you must compare the number of rows in the table to the number of rows masked. If the two are equal, then the masking operation is completed. But this does not indicate that the data masking operation was successful. If the masking script is incorrect, the masking operation may still complete but not produce the desired masked data outcome. To determine whether the data is properly masked, you must perform an audit of a statistical data sample.

Monitoring your Masking Jobs

Once a masking job has been created and started, you can monitor its progress by navigating to the Monitor tab or by clicking on the name of the masking job on any screen. The monitoring tab shows you a list of executed masking jobs, their progress as well as their current status. To get even more detail on the progress of an individual masking job, click on the Job Name.

Status

Status for a masking job refers to the job completion state. There are four statuses for a job:

- **Created:** means that a user has configured this masking job but it has never been executed.

- **Success**: means that the job has completed running as the user has defined. If the job encountered non-conformant data patterns while applying the specified masking algorithms

- **Failure or Errors:** means that the job failed before completion and that not all designated data was masked.

- **Running:** means that the masking job is currently in the process of being executed.

- **Queued:** means that the masking job is currently waiting to be executed until there are enough resources to be run.

> ℹ️ **Periodic Auditing of Masked Data**
>
> Please note that Success does not necessarily mean that all data has been masked (for example, if non-conformant data was encountered or if the user misconfigured the masking job and used the wrong algorithm). It is very important that an audit of the resulting masking data is periodically performed.

Progress

Progress refers to how much of the job as configured has been proceeded. Progress is measured with a range of 0% to 100%. Please note that there are several known bugs in the progress bar that results in lags or an inaccurate %. We recommend not using the progress bar as a measure of whether or not a job has been completed but instead relying on the Job Status.

Queue Position

Queue position refers to the job's numerical order of when it will be dequeued and run, relative to other queued jobs. If a job is not in the queue, it will not have a queue position.

Monitoring a Single Job

In addition to viewing high-level stats about the status/progress of all your jobs, you can also deep dive into each job to get more details. By clicking the name of the masking job, you will be redirected to a screen with more granular information including; environment name, connector name, job start time, previous run time, number of tables defined in the job, number of jobs tables masked, number of tables to be masked, the type of job, the total time the job has taken, rows remaining to mask, rows masked, number of streams, etc.

Environments	**Monitor**	Settings	Admin	Audit

Home > Monitor > Completed

Monitor

MASK_JOB

0 Jobs Running

100%	⭐ SUCCESS

Environment	Start Time	06:34:36	Total Time Taken	00:00:11
Test_MASK	Previous Run Time		Masking Report	Download Report
Job ID	Total # of Tables	3	Masking Inventory Report	Download Report
3	Tables Masked	3	Rows Remaining	0
Execution ID	Tables with Nonconforming Data ⓘ	0	Rows Masked	5
2	Tables to be Masked	0	Columns with Nonconforming Data ⓘ	0
CM Connection	Job Type	Mask	Streams	1
table			Updates Running	1
Source / Target			Repository	POSTGRESQL
- / biscuit				

Completed	Processing	Waiting

Completed

3 Complete **3** Total Tables

Name	Progress	Time	Rows Per Min	Rows Masked	Rows Remaining	Columns with Nonconforming Data
DBVERIFICATION_TABLE	100% ⭐	0d 0h 0m	326	4	0	0
Foo1	100% ⭐	0d 0h 0m	56	1	0	0
Foo2	100% ⭐	0d 0h 0m	0	0	0	0

Environments | Monitor | Settings | Admin | Audit

D Ξ L P H I X

In addition to seeing this additional information about each masking job, you can look into the status/progress of each table/file defined in the masking job. Each table/file will be separated into 1 of 3 tabs:

- **Completed:** The Completed tab shows which tables or files the job has completed and includes information such as the rows masked per minute, rows masked, and rows remaining.

- **Processing:** The Processing tab will include information on the tables or files the job is currently processing.

- **Waiting:** The Waiting tab shows us which table or files are waiting to be processed.

Displaying Non-conformant Data

When non-conformant data is encountered by a masking job, the job will either Fail or Succeed with a warning, depending on how the algorithms associated with the ruleset for the job are configured. As depicted in the screenshot, the non-conformant data can be accessed via the **Completed** tab on the Monitor page for the job, which can be accessed by clicking on the Job name from the Environment Overview page. In the main body of the Monitor page, a summary of the **Tables with Nonconforming Data** and **Columns with Nonconforming Data** is reported. Further details on the non-conformant data encountered can be accessed by clicking the Success or Fail icon next to each table or file listed in the **Completed** tab.

⚠ **Success Report - testdata_XML**

NONCONFORMING DATA   ⊘ **Learn More**

Event	Cause	Approximate Row Count	Description
UNMASKED_DATA	PATTERN_MATCH_FAILURE	1000	Column RCHARS64_T1_0 contained nonconforming data that was not masked by algorithm DateShiftVariable The top nonconforming data samples were: LLLLL LLLLLL LLLL LLLLLLL LLLLLLLLLLL LLLLLLLLL LLLLLLLL

1_27_testdata_XML.txt

2019/03/12 21:04:04 - testdata_XML - Loading transformation from XML file [/var/delphix/masking/output/Test/DMSApplicator/Oracle/1/KETTLE_MASK_XML_1_testdata_XML_27.xml]
2019/03/12 21:04:04 - testdata_XML - Using legacy execution engine
2019/03/12 21:04:04 - KETTLE_MASK_XML_1_testdata_XML_27 - Dispatching started for transformation [KETTLE_MASK_XML_1_testdata_XML_27]
2019/03/12 21:04:05 - Table input.0 - Finished reading query, closing connection.
2019/03/12 21:04:05 - Select values.0 - Finished processing (I=0, O=0, R=1000, W=1000, U=0, E=0)
2019/03/12 21:04:05 - Get All Lookups Values.0 - Finished processing (I=0, O=0, R=1000, W=1000, U=0, E=0)
2019/03/12 21:04:05 - Table input.0 - Finished processing (I=1000, O=0, R=0, W=1000, U=0, E=0)
2019/03/12 21:04:06 - User Defined Java Class.0 - Finished processing (I=0, O=0, R=1000, W=1000, U=0, E=0)
2019/03/12 21:04:06 - SelectValues_MetaData.0 - Finished processing (I=0, O=0, R=1000, W=1000, U=0, E=0)
2019/03/12 21:04:06 - String Cut.0 - Finished processing (I=0, O=0, R=1000, W=1000, U=0, E=0)
2019/03/12 21:04:07 - Update.0 - Finished processing (I=1000, O=0, R=1000, W=1000, U=1000, E=0)

The non-conformant data events are displayed followed by the masking log for the table or file. If there were no non-conformant data events, "None" is displayed under **NONCONFORMING DATA**, otherwise, for each type of non-conformant data, a row will be displayed with the following information:

- **Event type**: either JOB_ABORTED or UNMASKED_DATA if the job was not aborted.

- **Cause**: always PATTERN_MATCH_FAILURE.

- **Approximate Row Count**: approximate number of rows with non-conformant data (at least within an order of magnitude).

- **Description**: details the name of the column or field with non-

Interpreting Samples of Non-conformant Data Patterns

Each character in the non-conformant data is sampled per its Unicode Character Property.

- N for digits

- L for letters

- M for marks

- P for punctuation

- S for symbols

- Z for separator

- O for other

- U for unknown

Tracking Non-conformant Data

> **Info**
>
> Please note that actual personal data is never displayed, only the samples (a.k.a. patterns) of non-conformant data are displayed on this page

Using the DataBase specific SQL query, it is possible to locate data corresponding to the non-conformant data sample. The table and column names can be found on the table report. In the example above, the table name is "testdata_XML" and the column name is "RCHARS64_T1_0".

> **Note**
>
> The pattern might be not an exact representation of the data in the field, but a part of the data. For instance, white spaces at the beginning or at the end of the data might be truncated.

Oracle DB specific example

Below are the Oracle character classes, used in the regular expression:

Character Class Syntax	Meaning
[:alnum:]	All alphanumeric characters
[:alpha:]	All alphabetic characters
[:blank:]	All blank space characters.
[:cntrl:]	All control characters (nonprinting)
[:digit:]	All numeric digits
[:graph:]	All [:punct:], [:upper:], [:lower:], and [:digit:] characters.
[:lower:]	All lowercase alphabetic characters
[:print:]	All printable characters
[:punct:]	All punctuation characters
[:space:]	All space characters (nonprinting)
[:upper:]	All uppercase alphabetic characters
[:xdigit:]	All valid hexadecimal characters

For the LLLLL sample in the example above, Oracle DB SQL query would look like:

```
SELECT RCHARS64_T1_0 FROM testdata_XML WHERE regexp_like(RCHARS64_T1_0, '[[:alpha:]]{5}');
```

For the LLLLZLLLLZLLLL sample, the Oracle DB SQL query would look like:

```
SELECT RCHARS64_T1_0 FROM testdata_XML WHERE regexp_like(RCHARS64_T1_0, '[[:alpha:]]{4}[[:space:]]
[[:alpha:]]{3}[[:space:]][[:alpha:]]{4}');
```

Limitation for the Multi-Column Extensible Algorithm

If a Non-conformant data pattern is encountered - it is displayed for all the masked columns of the MC Algorithm, not only for the column where that event has occurred. In that case, the manual analysis of the error message will be required to find the actual column(s) with the Non-conformant data.

Masking Job Wizard

The Delphix Masking job wizard enables users to create and modify masking jobs. While the wizard facilitates a number of workflows and operations, more advanced functionality and finer control of features are available directly in the masking application. The Job Wizard currently functions only with certain data platforms, but these constraints do not apply when working directly in the masking application.

Supported Data Platforms

The following data platforms are currently supported from within the Job Wizard: - Oracle Database - RDS Oracle Database - MSSQL Server Database - Sybase Database

This restricted list only affects your use of the wizard; an expanded number of platforms are supported directly in the masking application. Some operations within the Job Wizard are also limited. See below for details.

Supported Operations

While creating a masking job in the Job Wizard, you are able to do the following:

- Create a new application or use an existing application

- Create a new environment or use an existing environment

- Create a new connector

- Create a new rule set

- Update inventory

- Create a masking job

- Update a masking job

- Change the connector for an existing job

- Change the rule set for an existing connector

- Run a newly created job immediately

- Run an updated job immediately after the update

> ✏️ **Note**
>
> Operations marked with an asterisk are limited in the Job Wizard but fully supported in the main application.

What is Not Supported in the Wizard

The following data platforms and operations are not supported in the Job Wizard. To access additional functionality, use the main masking application.

Unsupported Data Types

The following data types are supported when using the main masking application but are not currently supported in the Job Wizard:

- DB2 Database

- PostgreSQL Database

- Generic Database

- Delimited File

- Excel Sheet File

- Fixed File

- Mainframe Data Set

- XML File

Unsupported Operations

The following operations are not yet supported from within the Job Wizard:

- Creating any connector or rule set for an unsupported data type

- Deleting any application, environment, connector, rule set, or masking job

- Importing or exporting any object

- Updating an environment

- Creating a connector using Advanced mode

- Updating a connector

- Updating a rule set

- Creating a job for an unsupported data type

- Modifying a job for an unsupported data type

- Monitoring running jobs

- Creating, editing, deleting, or running any Profile jobs

Opening the Masking Job Wizard

When you first login to masking, the welcome screen offers a link to learn more or begin masking immediately. To open the Job Wizard, click Run on the welcome page.

✕

Welcome to Delphix Masking

If this is your first time using masking, here are some tools to get you started.

Read the Quick Start Guide	Watch an instructional video	Run the 'Create Job' Wizard
Read	**Watch**	**Run**
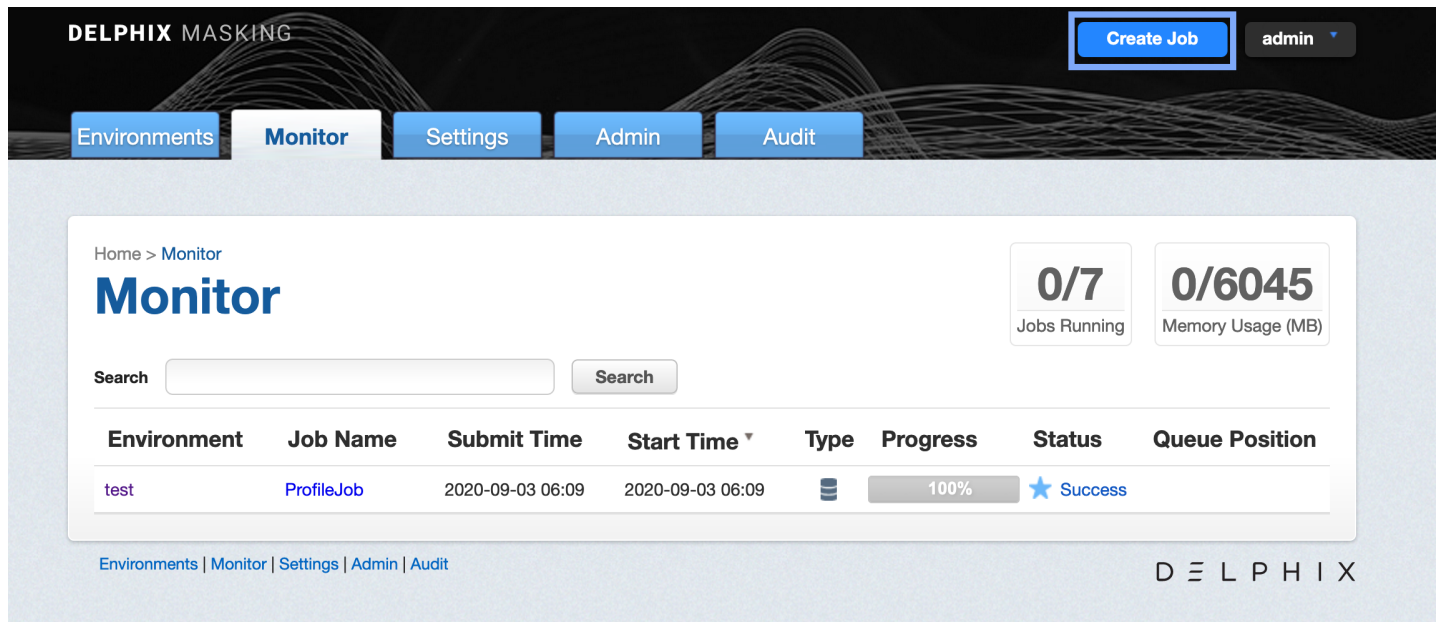
Notes

Do not run masking on your production data - always work with a copy.
You can launch the "Create Job" wizard by using the Job Wizard button in the application.

☐ Don't show again   D Ξ L P H I X

To use the Job Wizard from the masking application, click the Create Job button in the upper right-hand corner, as highlighted in the screenshot below.

DELPHIX MASKING   **Create Job**   admin ▾

Environments   **Monitor**   Settings   Admin   Audit

Home > Monitor

Monitor

0/7
Jobs Running

0/6045
Memory Usage (MB)

Search [   ]   Search

Environment	Job Name	Submit Time	Start Time ▾	Type	Progress	Status	Queue Position
test	ProfileJob	2020-09-03 06:09	2020-09-03 06:09	🗄	100%	★ Success	

Environments | Monitor | Settings | Admin | Audit   D Ξ L P H I X

> ✏️ **Note**
>
> Only administrators or users with the following privileges can see the Create Job button.
>
> - Environment: View, Add, and Update
> - Connection: View and Add
> - Rule Set: View and Add
> - Inventory: View and Update
> - Profile Job: View, Add, and Run
> - Masking Job: View, Add, Update, and Run
> - Inventory Report: View

Creating a New Masking Job

The Job Wizard makes creating a new masking job much easier by guiding you through the process. You can create new objects or choose to use existing ones that have already been defined. When creating a new masking job, the Job Wizard follows this sequence:
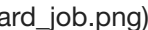
- Job Naming
- Application/Environment Selection
- Connection Selection
- Rule Set Selection
- Inventory Selection
- Summary Page

You can navigate back and forth through the pages of the Job Wizard.

> ✏️ **Note**
>
> If the product times out due to long inactivity, you will need to start over.

To create a new masking Job using the new Job Wizard, follow the procedure below:

1. Log into your Delphix Masking Engine and from the Welcome screen select Run.

2. Select the New radio button and enter a name for your Masking job. ![] (/media/wizard_job.png)

3. Click Next.

4. From the drop-down menu select an Application and Environment. If none exist use the Add button to add one. ![] (/media/wizard_job_env.png)

5. Click Next.

6. Select a Connector from the drop-down menu. If none exists select the Add button, then use the Add Connector dialog to add a new connector. The Job Wizard only supports the following Connector types:

   - Database - MS SQL

   - Database - Oracle

   - Database - RDS Oracle

   - Database - Sybase ![] (/media/wizard_job_connection.png)

7. Click Next.

8. On the Rule Set screen select an existing Rule Set or create a new one by clicking the Add button. ![] (/media/wizard_job_rule_set.png)

9. Click Next.

10. From the Inventory screen select how your data will be masked. In the screenshot below we are masking subscriber last names. ![] (/media/wizard_job_inventory.png)

11. Click Next.

12. The final screen of the Job Wizard displays a Summary of your selections. ![] (/media/wizard_job_summary.png)

13. Clicking Run Masking Job Now and go to Monitor progress, saves your job, and runs it immediately. Save Job allows you to save your job and run it at a later date. Note: Selecting this option means your data will not be masked until you run the job.

When Objects Are Saved

Application, environment, connector, and Rule Set objects are created and persist after you click the Add button and see a success message. If you cancel the Job Wizard before completing the job setup, the objects you created will be saved, and they will be available for use the next time you launch the Job Wizard.

The Inventory definition is saved when you change the selection of a table or column, or when another View filter is applied.

The masking job is saved when you click either Save Job or Run Masking Job Now and go to Monitor progress and a success message is returned on the Summary screen.

Updating an Existing Masking Job

You can use the Job Wizard to modify any masking job that targets a supported data type.

1. On the Job screen of the Job Wizard, select Modify Existing

2. From the list of available jobs select the one you want to modify. This list only shows jobs that are supported by the wizard. You can filter the job list by selecting the filter icon.

3. Once you select a job, you can change the following as part of the Modify flow:

   - Change/create a new Connector

   - Change/create a new Rule Set

   - Update inventory

- Save or run the modified job

You cannot alter application and environment settings as part of the Modify flow, but you can do so in the main masking application.

Running and Stopping Jobs from the Environment Overview Screen

To run or rerun a job from the Environment Overview screen:

- Click the Run icon (play icon) in the Action column for the desired job.

The Run icon changes to a Stop icon while the job is running. When the job is complete, the Status changes.

To stop a running job from the Environment Overview screen:

1. Locate the job you want to stop.

2. In the job's Action column, click the Stop icon.

3. A popup appears asking, "Are you sure you want to stop job?" Click OK.

4. When the job has been stopped, its status changes.

5. After the job completes successfully, return to the Inventory screen and check that the Domain and Method populated automatically for sensitive data. Sample screenshot below.
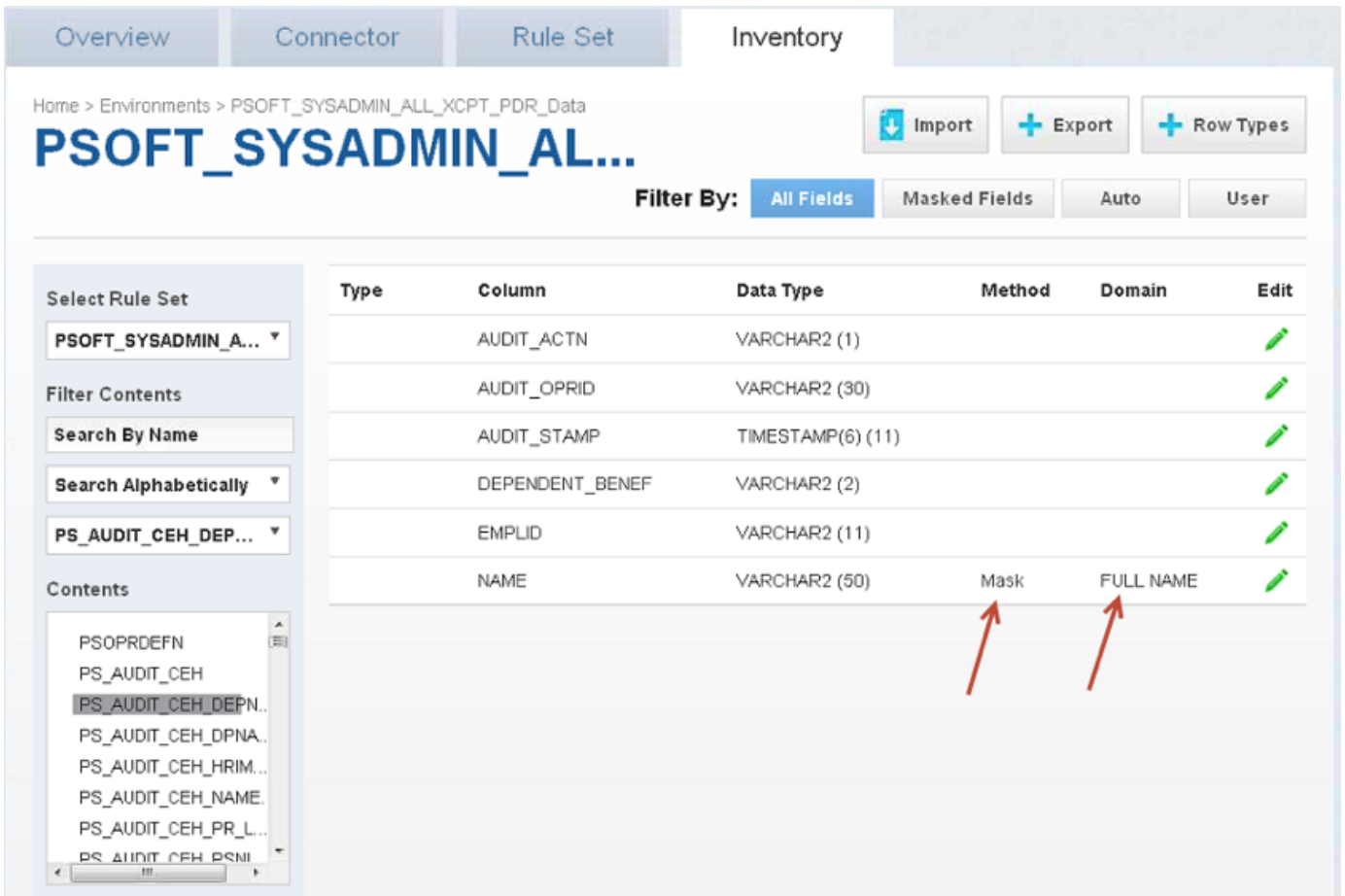
Builtin Driver Supports

Introduction

In 6.0.11.0, Delphix introduced the first built-in driver support plugin for the Oracle database platform.

The native connector types with a built-in driver support plugin are:

Native Connector Type	Release
Oracle	6.0.11.0
MSSQL	6.0.12.0

The built-in driver support plugins replace and improves upon the native connector database masking options of Disable Constraints, Drop Indexes, and Disable Triggers that have long had issues with functionality and negatively affecting job performance. Delphix has implemented the built-in driver support plugin for native connectors with Disable Constraints, Drop Indexes, and Disable Triggers tasks using the Driver Support Plugin Framework released in 6.0.9.0. These optimizations apply to masking, reidentification, and tokenization jobs where these tasks are enabled.

For details on how to enable/disable these tasks on supported native connector jobs using the new Driver Support Plugin Framework, see API Calls for Managing Masking Job Driver Support Tasks.

To retrieve information about job failures due to driver support task failures, an execution event will be raised and is accessible via the `GET /execution-events` endpoint: 1. `eventType` - `DRIVER_SUPPORT_TASK_FAILURE` 2. `exceptionDetail` - Error message about the task failure that will typically include the error code that is specific to the database platform

Oracle

For details on usage and known limitations of the Oracle Disable Constraints, Drop Indexes, and Disable Triggers driver support tasks, see Oracle Built-in Driver Support Plugin.

MSSQL

For details on usage and known limitations of the MSSQL Disable Constraints, Drop Indexes, and Disable Triggers driver support tasks, see MSSQL Built-in Driver Support Plugin.

Masked Provisioning

Configuring Virtualization Service for Masked Provisioning

Introduction

During the VDB provisioning process, the Virtualization Engine can optionally run a masking job from a Delphix Masking engine on the VDB. Use these instructions to customize the host address, port number, and/or login credentials that the Virtualization Engine will use to contact the Masking Engine.

> ⚠ **Important Validation Notices**
>
> When configuring masked provisioning, ensure that the versions of the Virtualization Engine and Masking Engine are compatible. See the compatibility matrix.
>
> Old versions of the serviceconfig or any information associated with them are not tracked. In particular, if you have been using the local masking service or a remote service and then change to a new remote service Delphix will start throwing out any old job information on the next masking job/fetch or GUI reload. Users should not rely on that information being preserved through serviceconfig updates.
>
> Delphix does not validate network availability between the two engines or any other hosts that both engines might want to communicate with. The state or availability of either host is not checked, if either host becomes unduly slow, congested, or unresponsive Delphix will not be able to issue compelling warnings regarding those issues.

Instructions

Use these instructions to customize the host address, port number, and/or login credentials that the Virtualization Engine will use to contact the Masking Engine.

> ✏ **Note**
>
> This does not alter the Delphix Masking Engine UI port. It is specific to coordinating communication between the Virtualization Engine and a Masking Engine about available masking jobs and job results.

To change the Virtualization Engine's connection details for its Masking Engine:

1. Using a shell, login to the **CLI** using:

   - On 5.2 and earlier releases: **delphix_admin**.

   - On 5.3 and later releases: **admin**.

2. At the **CLI** root prompt, type **maskingjob**.

3. At the **maskingjob** prompt, type **serviceconfig**.

4. To list service configurations, type **ls**.

5. At the serviceconfig, type **select `MASKING_SERVICE_CONFIG-1**.

6. To view the configurations, type **ls**.

7. With this service config selected, enter **update**.

8. In the update mode, use the **set** command to modify the configuration. For example, type **set port=[YOUR DESIRED PORT NUMBER]** to change the port number.

9. Commit the change by typing **commit**.

10. Type **ls** to confirm the configurations.

11. Type **exit** to exit the CLI.

Provision Masked VDBs

Masked virtual databases (VDBs) function just like normal VDBs. The only distinction is that the data they contain has been masked by a masking job. Masked VDBs can be replicated to a separate Delphix Engine (in non-prod) without sending the original data that was obfuscated during masking using a process called Selective Data Distribution (SDD). This topic describes how to work with masked VDBs.

Prerequisites

Before attempting to create a Masked VDB, you should be familiar with both Delphix Virtualization and Delphix Masking concepts and workflows.

Restrictions

- A single masking job cannot be assigned to multiple VDBs simultaneously. If you are using the same masking ruleset on multiple VDBs, be sure to create a unique job for each VDB to avoid any issues with provisioning or refreshing.

- Provisioning or refreshing masked VDBs is only supported for Oracle, MS SQL Server, and Sybase. Provisioning or refreshing other types of masked VDBs such as DB2 are not support.

- You cannot apply additional masking jobs to a masked VDB or its children.

- If a masking job has been applied to a VDB, you cannot create an unmasked snapshot of that VDB.

- Masking must take place during the process of provisioning a VDB. If an existing VDB has not had a masking job applied to it, then you cannot mask that particular VDB at any point in the future. All the data within the VDB and its parents will be accessible if it is replicated using SDD.

- When selecting a connector to use for Masked Provisioning, a "basic" connector must be used **unless** you are masking an Oracle Pluggable Database (PDB), in which case an "advanced" connector must be used.

- Only in-place masking jobs can be selected.

- Masked Provisioning is supported on Oracle RAC only when used with "script-based masking" and not when a masking job is used for SDD.

Identifying and Navigating to Masked VDBs

Masked VDBs appear in the Virtualization Engine's Datasets pane, just like regular VDBs. They are most obviously identified by the different icons used to represent them. In addition, a masked VDBs Configuration tab will contain information about the masking job that you applied to it. Generally, anything you can do with an unmasked VDB is also possible with a masked VDB. ![] (/media/masked_vdb.png)

Provisioning Masked VDBs

- In the Virtualization Engine, associate a masking job with a dSource.

- Use the dSource provision wizard to provision a VDB with a masking job.

Associating a Masking Job with the dSource

To provision a masked VDB, you must first indicate that the masking job you are using is complete and applicable to a particular database. You do this by associating the masking job with a dSource.

1. In the **Datasets** panel on the left-hand side of the screen, click the dSource to which the masking job is applicable and with which it will be associated.

2. Click the **Configuration** tab.

3. Click the **Masking** tab. ![] (/media/masking_tab.png)

4. Click the **pencil** icon to edit. All masking jobs on this Delphix Engine that have not been associated with another dSource will be listed on the right-hand side.

5. Select the **job** you want to associate with this dSource.

6. Click the **green checkmark** to confirm. ![] (/media/mask_job.png)

7. Repeat for any other jobs that you want to associate with this dSource at this time.

The Delphix Engine now considers this masking job to be applicable to this dSource and ready for use. When provisioning from snapshots of this dSource, this masking job will now be available.

> ✏️ **Note**
>
> Masking jobs can also be associated with virtual sources in addition to dSources.
>
> A masking job must be Multi-Tenant for creating a masked VDB. The Multi-Tenant option allows existing rulesets to be reused to mask identical schemas via different connectors. The connector can be selected at job execution time.

Provisioning a Masked VDB using the dSource Provisioning Wizard

The steps required to provision a masked VDB are almost identical to the steps required to provision an unmasked VDB. Once you have created a masked VDB, you cannot un-mask it, nor can you alter which masking job it uses. All snapshots in the VDBs TimeFlow will always be masked using the masking method that you selected when you provisioned the masked VDB.

1. In the **Datasets** panel on the left-hand side of the screen, select the dSource.

2. Click the **TimeFlow** tab.

3. Click **Provision VDB** icon.

4. Review the information for Installation Home, Database Unique Name, SID, and Database Name. Edit as necessary.

5. Review the Mount Base and Environment User. Edit as necessary.

   - If you want to use login credentials on the target environment that are different from the login credentials associated with the Environment User, select Specify Privileged Credentials.

6. Click **Next**.

7. If necessary, edit the **Target Group** for the VDB.

8. Select the **None** option for the Snapshot Policy for the VDB.

   - **Snapshot Policy Selection**: For almost all use cases involving Masked VDBs, a Snapshot Policy of None is appropriate. Using a Snapshot Policy in conjunction with SDD can result in the leak of sensitive data.

9. Click **Next**.

10. Click **Mask this VDB**. You will be presented with two options to mask this VDB:

    - Select an existing masking job: Choose this option if you want to mask using a preconfigured Masking Job. Only masking jobs that have been associated with the parent dSource will be available.

    - **Selecting Unique Masking Jobs**: If you are using the same masking ruleset on multiple VDBs, be sure to create a unique job for each VDB to avoid any issues when provisioning or refreshing. ![] (/media/mask_rule_set.png)

    - Masking using scripts(s): Alternatively, you may define some Configure Clone scripts in the Hooks step to perform masking.

    - **Defining Configure Clone Hooks to Mask VDB**: If you choose to mask using script(s), you must define the Configure Clone hooks to run masking jobs yourself. If you don't define any Configure Clone hooks in the Hooks step, the data will be marked as masked, but it will not be masked. ![] (/media/mask_hooks.png)

11. Click **Next**.

12. Specify any **Pre or Post Scripts** that should be used during the provisioning process. If the VDB was configured before running the masking job using scripts that impact either user access or the database schema, those same scripts should also be used here. Be sure to define the **Configure Clone** hooks to run the masking job if you choose to mask using script(s) in the Masking step. ![] (/media/hooks.png)

13. Click **Next**.

14. Click **Submit**.

If you click Actions in the upper right-hand corner, the Actions sidebar will appear and list an action indicating that masking is running. You can verify this and monitor progress by going to the Masking Engine page and clicking the Monitor tab.

![] (/media/monitor.png)

> ✏️ **Note**
>
> Once you have created a masked VDB, you can provision its masked data to create additional VDBs, in the same way, that you can provision normal VDBs. Since the parent masked VDB contains masked data, child VDBs will only have masked data. This is a great way to distribute multiple independent copies of masked data that is both time and space-efficient.

Refresh a Masked VDB

You refresh a masked VDB in exactly the same way as you refresh a normal VDB. As with provisioning a masked VDB, the masking job will be run during the refresh process.

1. Login to the Delphix Management application.

2. Click Manage.

3. Select Datasets.

4. Select the VDB you want to refresh.

5. Click the Refresh VDB button (2 circular arrows).

6. Select More Accurate and Next.

7. Select the desired refresh point snapshot or click the eye icon to choose the latest available range, A point in time, or An SCN to refresh from.

8. Click Next.

9. Click Submit to confirm.

10. Click the Actions link to watch the progress of the refresh job.

11. To see when the VDB was last refreshed/provisioned, check the Time Point on the Status page.

Disassociating a Masking Operation on a dSource

If a masking job is found to be unsuitable or should be retired, you can disassociate it though the same database card that you used to associate it.

1. Deselect the job.

2. Click the green arrow to confirm. Note that this will only prevent the creation of new masked VDBs with this job. It will not alter existing masked VDBs in any way. When disassociating a job, review the existing masked VDBs and consider whether you need to delete or disable any of them.

Masked VDB Data Operations

The following data operations are available to masked VDBs:

- Rewind: Alter the database to contain masked data from a previous point in time.

- Refresh: Get new data from the parent dSouce and mask it.

- Disable: Turn off the database and remove it from the host system.

- Enable: Turn on the database and make it available on the host system.

Virtualization and Masking Engine Compatibility Matrix

Virtualization Engine Version	Masking Engine Version
5.0 releases	5.0 releases (minor versions do not need to match)

Virtualization Engine Version	Masking Engine Version
5.1 releases	5.1 releases (minor versions do not need to match)
5.2 releases	5.2 releases (minor versions do not need to match)
5.2.5.0 (or later 5.2 minor release)	5.2.5.0 (or later 5.2 minor release)
5.3.0.0 and later, including later major releases (e.g. 6.0)	5.3.0.0 and later, including later major releases (e.g. 6.0) and minor versions do not need to match

Managing Multiple Engines for Masking

Introduction

Your organization may have more than one masking engine, and in certain circumstances, it may want to coordinate the operation of those engines. In particular, there are two specific scenarios in which an organization could benefit from some level of interaction and orchestration between multiple masking engines.

Software Development Life Cycle (SDLC)

Using an SDLC process often requires setting up multiple masking engines, each for a different part of the cycle (Development, QA, Production).
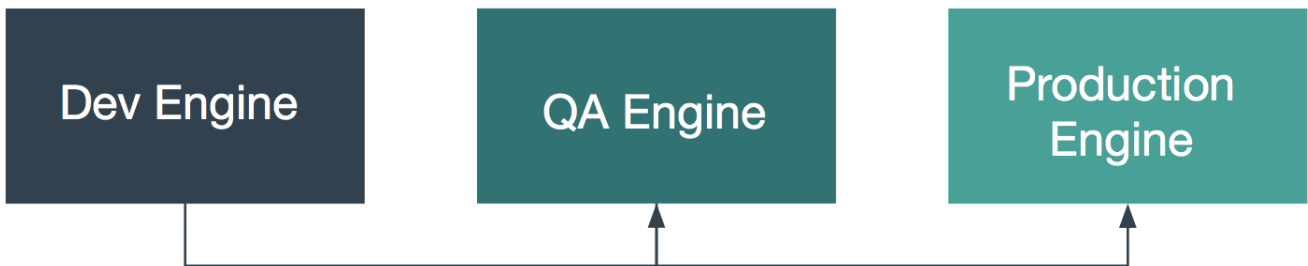
Horizontal Scale

For many organizations, the size of the profiling and masking workloads requires more than one production masking engine. These masking engines can be identical in configuration or be partially equivalent depending on the organization's needs.
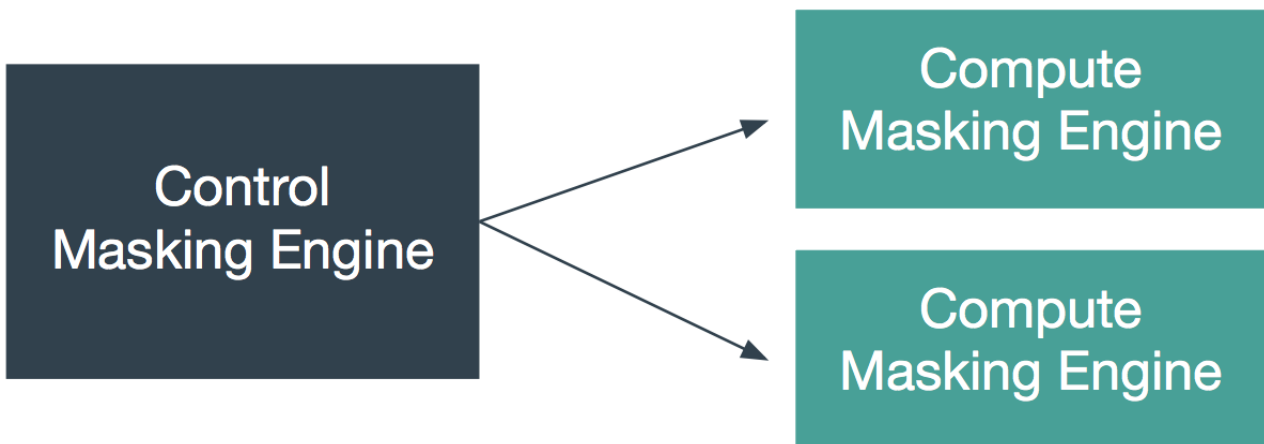
Best Practice Guide and Example Architectures for Synchronizing

Both of these use cases require various objects to be moved between masking engines, such as Connectors, Rule Sets, and more. Engine synchronization provides a general and flexible way to move the objects necessary to run an identical job on another engine. The following sections describe how to use the Masking APIs to accomplish this.

It is recommended that the syncable objects move in **only one direction**. That is, objects should be exported from one engine and imported into others but should not go in the other direction. This recommendation is primarily to simplify management of which objects exist on which engine.

For each of the scenarios above, an example architecture is described below. Note that the two architectures could be combined by having multiple production engines instead of a single one.

SDLC

The first architecture addresses the desire to author algorithms on one engine, to test and certify them on another, and finally to deploy them to a production engine. Here, algorithms are authored on the first engine, labeled "Dev Engine" in the diagram below. When the developer is satisfied, the algorithms are exported from the Dev Engine and imported to the QA Engine where they can be tested and certified. Finally, they are exported from the QA engine and imported to the production engine.

SDLC (Algorithm) Use Case
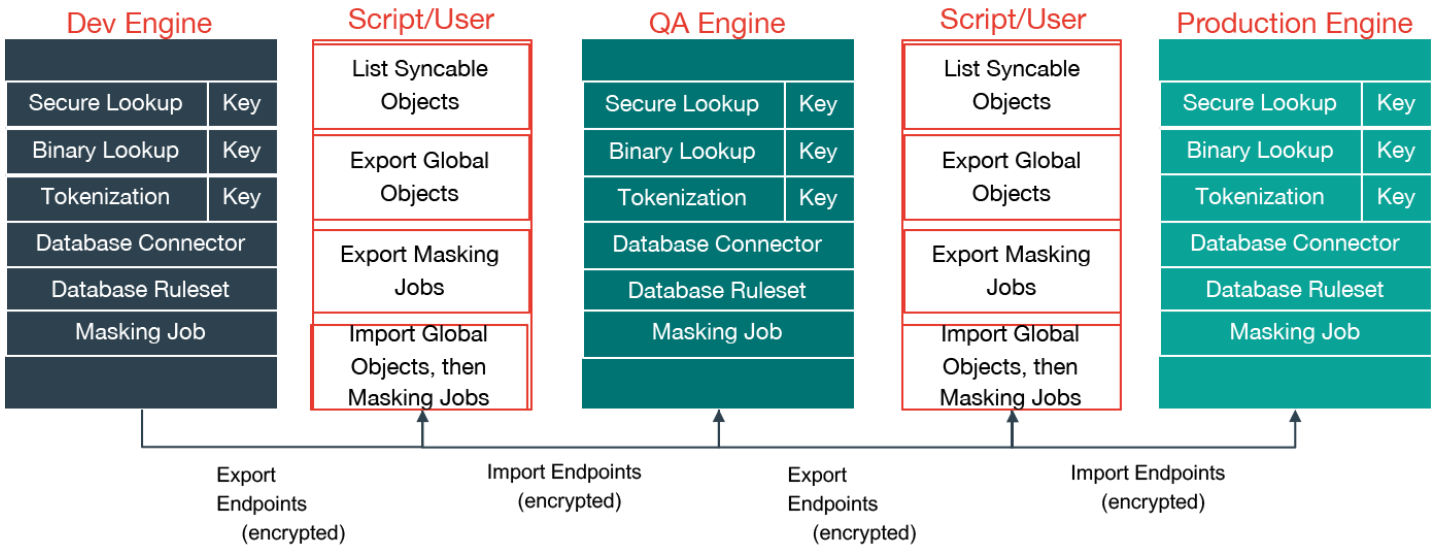
Horizontal Scale

The second architecture aims to address the problem of horizontal scale -- that is, achieving consistent masking across a large data estate by deploying multiple masking engines. In this architecture, syncable objects are authored on one engine, labeled "Control Masking Engine" in the diagram below. Those objects are then distributed to "Compute Masking Engines" using the engine synchronization APIs. The synchronized algorithms and masking jobs will produce the same masked output on all of the engines, thus enabling large data estates to be masked consistently.

Horizontal Scale Use Case

Control Masking Engine

Secure Lookup	Key
Binary Lookup	Key
Tokenization	Key
Database Connector	
Database Ruleset	
Masking Job	

Export Endpoints

(Encrypted)

Script/User

- List Syncable Objects
- Export Global Objects
- Export Masking Jobs
- Import Global Objects, then Masking Jobs

Import API Endpoints (Encrypted)

Import API Endpoints (Encrypted)

Compute Masking Engine

Secure Lookup	Key
Binary Lookup	Key
Tokenization	Key
Database Connector	
Database Ruleset	
Masking Job	

Compute Masking Engine

Secure Lookup	Key
Binary Lookup	Key
Tokenization	Key
Database Connector	
Database Ruleset	
Masking Job	

Sync Concepts

Syncable object

Syncable objects are external representations of masking engine objects that can be exported from one engine and imported into another. Sync currently supports exporting a subset of algorithms (see Algorithm Syncability for details), the encryption key and all the objects necessary for a job.

> ✏️ **Note**
>
> Sync does not currently support the following object(s):
>
> - Applications

> ⚠️ **Warning**
>
> Forward compatibility is not supported for engine sync, meaning that sync bundles from newer version engines may not import successfully into older version engines. If attempted, this could potentially result in an unexpected state or an error on the older version engine. However, backwards compatibility is supported and sync bundles from older version engines will import as expected into newer version engines, unless the sync bundle contains objects for a deprecated feature that no longer exists on the newer version engine.

Object Identifiers and Types

Sync uses object identifiers to name unique objects within the engine. The */syncable-objects* endpoint provides a list of all object identifiers for a particular object type.

The following object types are currently supported:

- ALGORITHM_PLUGIN
- APPLICATION_SETTINGS
- DATABASE_CONNECTOR
- DATABASE_RULESET
- DATASET_CONNECTOR
- DATASET_FORMAT
- DATASET_RULESET
- DOMAIN
- ENVIRONMENT
- FILE_CONNECTOR
- FILE_FORMAT

- FILE_RULESET

- GLOBAL_OBJECT

- JDBC_DRIVER

- KEY

- Certain algorithms:

  - BINARYLOOKUP

  - CLEANSING

  - DATE_SHIFT

  - LOOKUP

  - MAPPLET

  - MIN_MAX

  - REDACTION

  - SEGMENT

  - TOKENIZATION

- MASKING_JOB

- MOUNT_INFORMATION

- PROFILE_EXPRESSION

- PROFILE_JOB

- PROFILE_SET

- REIDENTIFICATION_JOB

- TOKENIZATION_JOB

- USER_ALGORITHM

The following lists the object types that are simply for the purpose of referencing a particular state of the exported object. These are not meant to be exported by request. The functions of these are further explained in the latter sections.

- ALGORITHM_REFERENCE

- DOMAIN_REFERENCE

- PROFILE_EXPRESSION_REFERENCE

- PROFILE_SET_REFERENCE

- SOURCE_DATABASE_CONNECTOR

- SOURCE_FILE_CONNECTOR

Dependencies

> ✏️ **Note**
>
> When exporting masking objects, a single export cannot contain multiple objects with the same name (e.g., two connectors with the same name).

Most objects within the Masking Engine are compositional. In order to properly capture the behavior of a syncable object, you must export its dependencies along with the object itself. Fortunately, all the necessary dependencies are exported along with the object you request; thus, it is not something you need to keep track of and worry about.

Syncable Object dependencies relationship
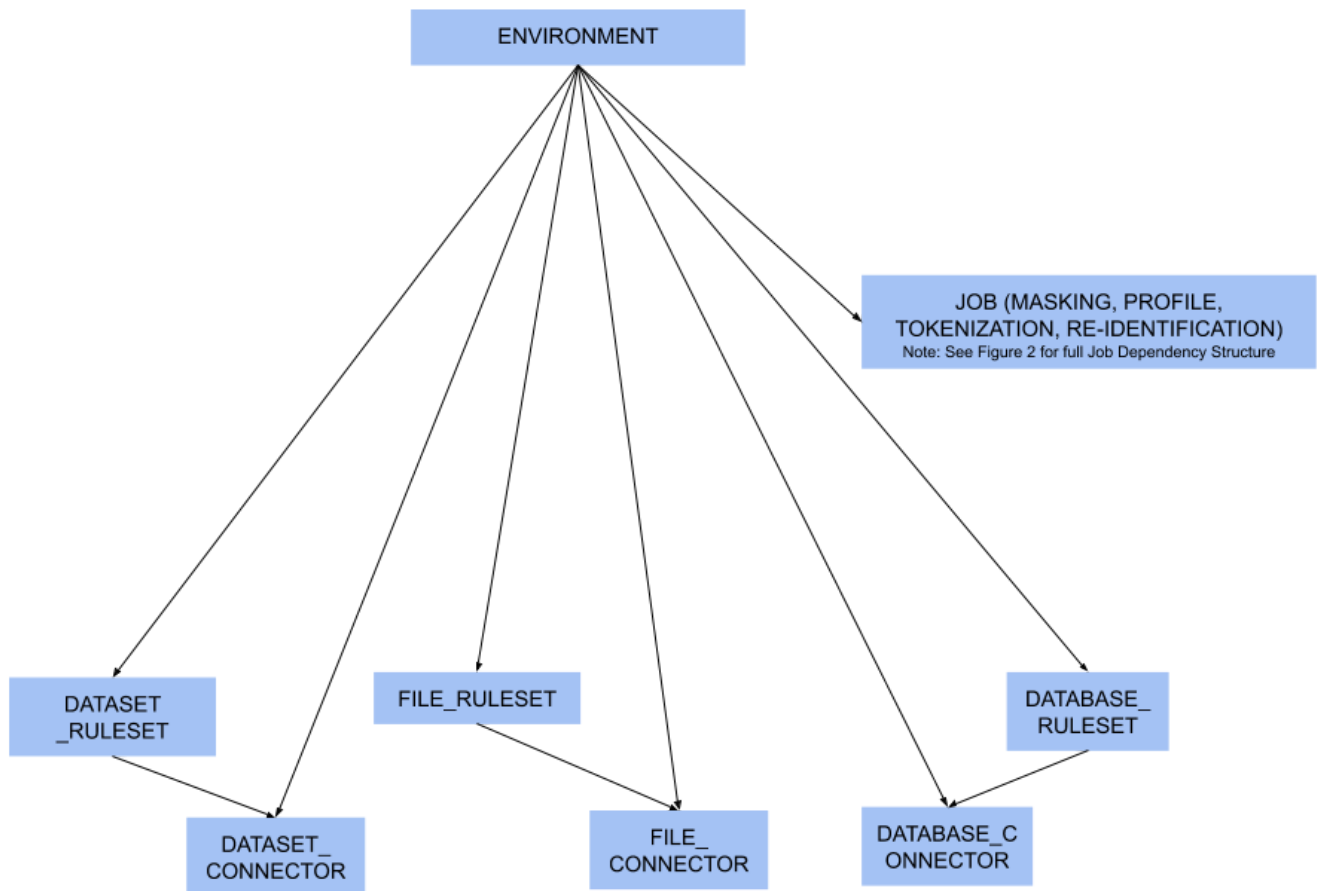
Figure 1 - environment dependencies

> ✏️ **Note**
>
> While rulesets and connectors are dependencies for Jobs (see Figure 2), you may also have rulesets and connectors that are not assigned to a job. In this case, they are considered to be direct dependencies for an environment.
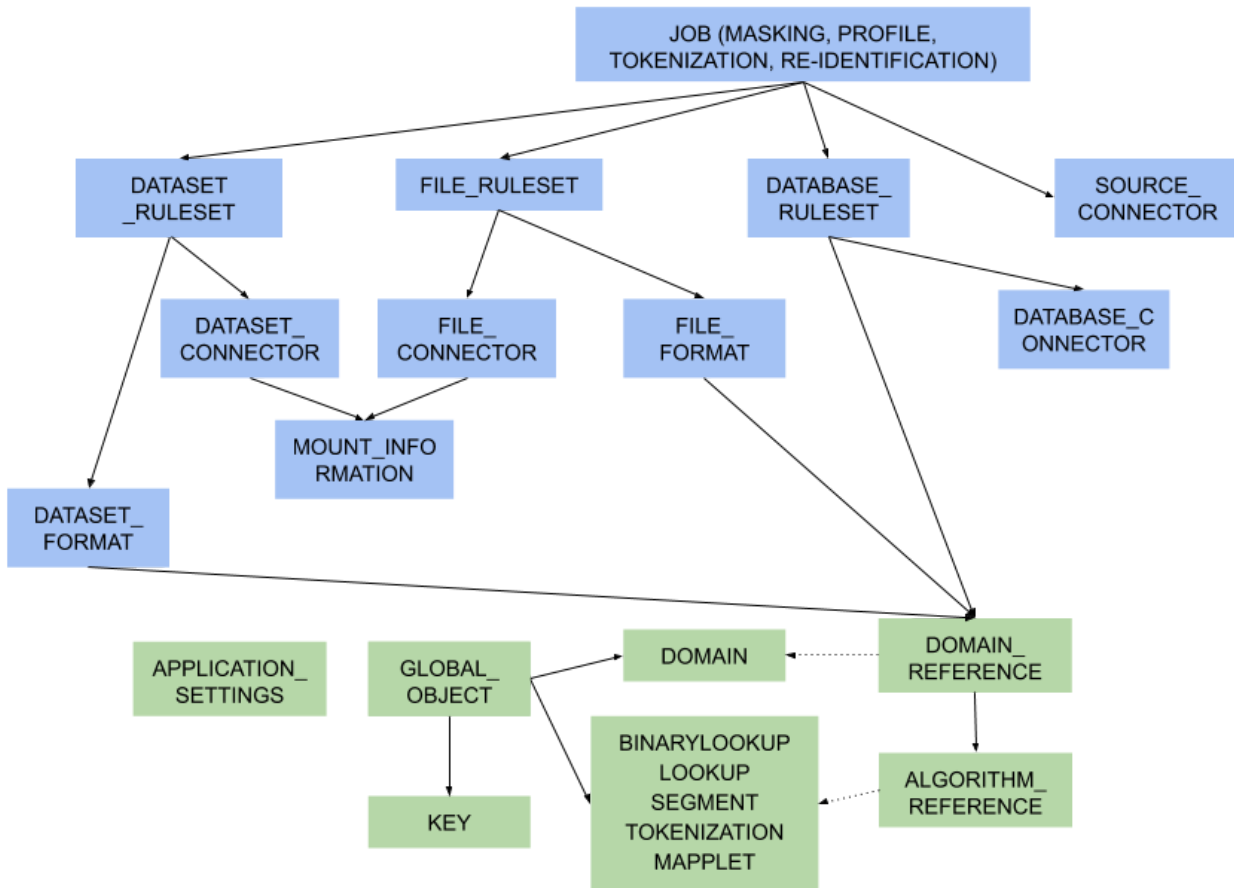
Figure 2 - object dependencies

> ✏️ **Note**
>
> Green represents global objects (objects that are central to the entire engine), and blue represents objects that need to be a part of an environment

Object Revision Tracking

The revision hash is used to help you determine whether the behavior of a syncable object is the same between engines. Because objects within the Masking Engine are compositional, the behavior of an object is influenced by all of its dependencies. When a syncable object is listed or exported, the Masking Engine computes a revision_hash, which uniquely identifies the object's behavior.

The revision_hash is a SHA1 hash that represents that object's state, as well as the state of all objects it depends on. If two objects have the same revision hash, it is safe to assume that the behavior of the objects is the same. However, it is possible for two objects to have the same behavior but have divergent revision hashes. For example, you could have two lookup algorithms with the same name, lookup file, and key, and they do not necessarily guarantee to have the same revision hash.

> ✏️ **Note**
>
> The revision_hash does not change when the password or the ssh key for the FILE_CONNECTOR, DATASET_CONNECTOR or DATABASE_CONNECTOR is updated. This is intentionally done because we do not export the password or the ssh key for security purposes. This allows users to update the password after import without changing the revision_hash. If a user is **overriding** a connector that already has a password set, the import **does not** reset the password and will leave the current, pre-import value.

> ✏️ **Note**
>
> The revision_hash may change from version to version, and the hash comparison should be done only if both the source engine and the target engine are on the same version of the product. It is also not guaranteed to be the same between two engines at the same version if they are synced from an engine at some other version. E.g. There are three engines as follows
>
> | Engine | Version |
> |--------|---------|
> | A | 5.3.2.0 |
> | B | 5.3.3.0 |
> | C | 5.3.3.0 |
>
> If B and C are synced from A, then the revision_hash is not guaranteed to be the same between B and C.
>
> **Best Practice:** A -> B -> C.

Export Document

You can export one or more syncable objects that are listed in the */syncable-objects* endpoint. The export document will include the set of objects that you requested for export and all of their dependencies that are required to properly import those objects into another engine.

The export document is exported as an opaque blob. Do not edit it outside of the Masking Engine.

Security

In most cases, an export document contains all the state necessary to re-create each of its objects (see this note about connector objects for one exception). In some instances, users might consider an object to be sensitive. For example, an algorithm object contains all of the information needed to produce identical algorithm results on a different engine (the algorithm's secret key, etc.). If the algorithm is being used in a production environment, then users may consider the algorithm definition and any export document containing the algorithm to be sensitive information. Therefore, export document access control, transmission, and storage should all be considered with care.

Access Control

The Masking Engine only allows administrative users to make Sync API calls. When creating an administrative user account, keep in mind that the account owner will be able to access the Sync APIs to export and import objects. For this and other reasons, administrative accounts should only be created for trusted individuals.

Non-administrative accounts are not allowed to use the Sync APIs.

Transmission Security

An export document containing a sensitive object should only be transmitted over a secure channel. This applies to situations where the Masking Engine is one of the transmission endpoints and when it is not. For example, when uploading (downloading) an export document to (from) the Masking Engine, the Sync API calls, like all Masking API calls, should be performed over HTTPS. Similarly, if an export document is transferred from a user's laptop to a server, the export document should be transmitted securely.

Storage Security

An export document containing a sensitive object should be encrypted before it is stored persistently. Users are free to apply an encryption mechanism of their choosing to an export document. As a convenience, you can request that the export document be encrypted by the Masking Engine using a passphrase. The Masking Engine will encrypt the export document with 3DES using SHA1 (PBEWithSHA1AndDESede). Once the document is encrypted with the passphrase, the engine forgets the passphrase. You will need to provide the same passphrase during import to decrypt the document.

Digital Signature

In order to detect accidental or malicious modification of the export document, each document is digitally signed. If the export document does not match its expected digital signature, a Masking Engine will not import the document.

Overwrite

When an object to be imported has the same name as a currently existing object, importing it will cause the other object to be changed. Since this might not be intended, we offer a flag called force_overwrite. If force_overwrite is set to false and doing the import will change an existing object on the masking engine, we fail the import. This workflow is shown below.
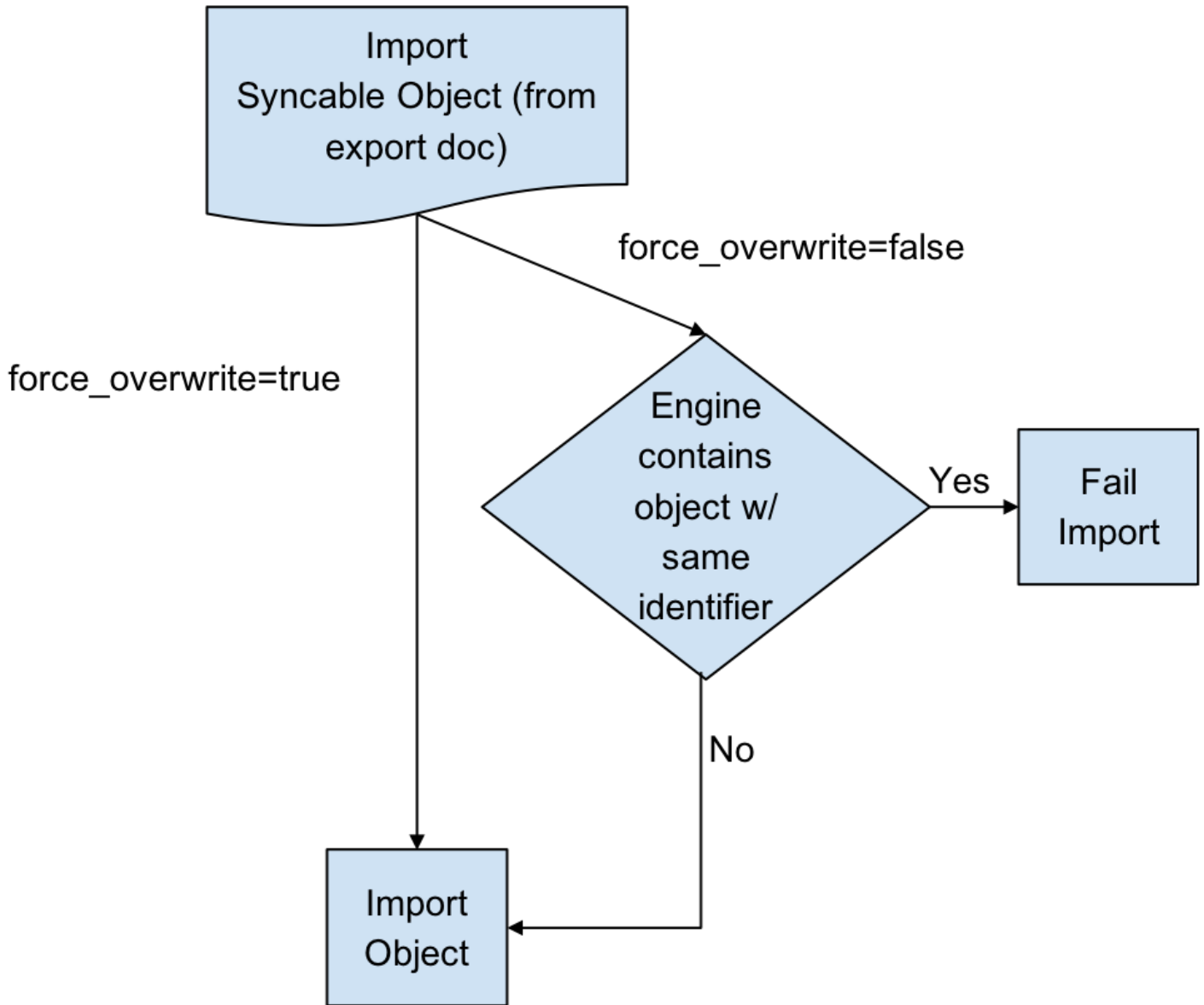
Attempting to Import Identical Objects

The Masking Engine checks for the existence of the same object contents during the import of an object. If it is determined that the engine and the document being imported contain the same content, a result of SUCCESS will be returned without repeating the work of a full import. For example, importing an entire ruleset with hundreds of thousands of tables can be quite time consuming, and this should not be repeated if the same object already exists. If the object content matches and we skip the full import we note this in the application log.

Below is an example of a log statement when an identical database connector was imported:

```
2017-07-19 10:17:06,075 [http-nio-exec-4] INFO
c.d.s.marshallers.SyncableMarshaller - Skipping import process for
{
"objectType": "DATABASE_CONNECTOR",
"id": {
"@type": "type.googleapis.com/IntegerIdentifier",
"id": 1
}
}, due to no discrepancy between the existing and importing object
```

Depending on the object type, some define an object by a String (name) and some by an Integer (object id). Objects that can have the same name in multiple environments, such as connectors, rulesets, and masking jobs, are exported based on a unique id associated with them. Global objects, which do not have overlapping names, are exported and identified based on their names. Something to note here is that objects exported based on their ids will overwrite the object with the *same name* rather than the same id. This means that for all importing objects, we define the identity of an object to be based on the name in the same environment.

For example, if I export a database connector named *testConnector* with the following export object metadata:

```
{
"objectIdentifier": {
"id": 5
},
"objectType": "DATABASE_CONNECTOR",
"revisionHash": "68eaffef400e426520a5fcbb683419db3be53317"
}
```

And then I import this object into some engine's environment with the following list of connectors:

id	connector name	more information
1	testConnector	...
5	otherConnector	...

testConnector of id 1 will be overwritten, instead of *otherConnector*.

Overwrite of the Encryption Key

The global encryption key is somewhat special in that it always exists. Specifying *force_overwrite=false* will always fail to import the encryption key unless the encryption key has been previously synchronized using *force_overwrite=true*.

Specifying *force_overwrite=true* will always overwrite the engine's encryption key with the contents of the encryption key in the export document.

Error handling

Export documents often have multiple objects to be imported at once. For example, when exporting a database ruleset, you will export both the database ruleset and the database connector since a ruleset depends on a connector.

The engine will import one object at a time, where the dependencies are imported first. If there is an error importing an object, the import process will abort and all objects that have successfully been imported during this request will get rolled back. For example, say you are importing objects A, B, and C. Import successfully imports A. During the import of B, the engine encounters an error. The import of A will roll back, and the import of C will never execute. This will leave the engine in a state identical to the one it was in prior to the failed import.

Concurrent Sync Operations

To prevent race conditions with concurrent imports and jobs running, we currently do not allow concurrent import operations. We also do not allow imports while masking jobs or exports are running. It is best to do imports when a machine is not running jobs or other exports in order to guarantee that the final state of each of those operations is as expected. If they are done at the same time, the operations will fail with relevant error messages.

Global Objects

GLOBAL_OBJECT is a syncable object type that is a collection of all syncable algorithms, ALGORITHM_PLUGIN(s), DOMAIN(s), JDBC_DRIVER(s), PROFILE_SET(s), PROFILE_EXPRESSION(s) and KEY (global key). This represents objects in the Masking Engine that are available across all environments, and are not a part of any specific environment. When a user requests to export GLOBAL_OBJECT, every syncable algorithm, profile set, profile expression and domain on the engine will be exported as the bundle. If a DOMAIN, PROFILE_SET, or PROFILE_EXPRESSION has a dependency on a non-syncable algorithm, such as Mapping, it will not be exported.

This separation was added because global objects 1) containing large lookup files are projected to be time-consuming and 2) are expected to be synchronized much less frequently than any masking job-related metadata. Examples on how to use it will be available in the Example User Workflow section.

References Objects

As mentioned in the *Global Objects* section, we expect the users to synchronize global objects and masking jobs at different frequencies. To avoid any unnecessary export of large algorithms, any objects (MASKING_JOB, PROFILE_JOB, DATABASE_RULESET, FILE_FORMAT and FILE_RULESET) that have dependencies on algorithms will export just the references to the objects by default. This way we check whether the necessary dependency exists on the importing engine by comparing the references; if not, we fail the import execution with an appropriate message. Domains, profile sets, and profile expressions are the exception to this. Exporting any of these objects will also export the full algorithm.

On-The-Fly Masking Jobs

By definition, On-The-Fly (OTF) masking jobs work with a source environment/connector and a target environment/connector, masking the data from the source connector into that of the target connector. With masking jobs, a target *environment_id* is always required to specify which environment to import the job and its target connector. In addition to the target *environment_id,* OTF masking jobs require the specification of a *source_environment_id* into which to import the source connector. The source connector is copied into the specified source environment (*source_environment_id*), and is represented by the SOURCE_DATABASE_CONNECTOR or

SOURCE_FILE_CONNECTOR for database and file masking jobs respectively in the export document. These source connectors are virtually identical to their DATABASE_CONNECTOR and FILE_CONNECTOR counterparts, but are represented differently in the OTF jobs to distinguish them from the target connector (i.e., DATABASE_CONNECTOR or FILE_CONNECTOR).

Circular Dependencies

It is possible to have a set of objects that end up depending on each other. This would be the case if a PROFILE_SET depended on a PROFILE_EXPRESSION that depended on a DOMAIN that depended on a REDACTION algorithm that depended on the original PROFILE_SET. The masking application will detect such scenarios on export and refuse to export such configurations.

This can be worked around by creating a second PROFILE_SET that contains PROFILE_EXPRESSIONS that do not depend on a DOMAIN that depends on a REDACTION algorithm. Simply ensure that the regular expressions are the same in the newly created PROFILE_EXPRESSIONs and assign the REDACTION algorithm to the new PROFILE_SET instead. The REDACTION algorithm will function the same but the dependency loop will have been broken.

Sync Endpoints

> **✏️ Note**
>
> When exporting masking objects, a single export cannot contain multiple objects with the same name (e.g., two connectors with the same name).

GET /syncable-objects

```
GET /syncable-objects[?object_type=<type>]
```

This endpoint lists all objects in an engine that are syncable and can be exported. Any object which can be exported can be imported into another engine. The endpoint takes an optional parameter to filter by a specific object type. Each object is listed with its revision_hash. Note that if a syncable object depends on a non-syncable object (e.g. DOMAIN using a mapping algorithm), it will say so in the "revisionHash" attribute, and will not be exportable.

Example CURL command using the object_type parameter to only retrieve the list of LOOKUP algorithm objects:

```
curl -X GET
--header 'Accept: application/json'
--header 'Authorization: 21c45f0e-82f4-4b04-9072-b49072986231'
'http://masking-engine.com/masking/api/syncable-objects?object_type=LOOKUP'
```

POST /export

This endpoint allows you to export one or more objects in batch fashion. The result of the export is an export document and a set of metadata that describes what was exported. You are expected to specify which objects to export by copying their object identifiers from the /syncable-objects endpoint.

> **✏️ Note**
>
> - The Sync POST /export API will timeout after 3 minutes.
> - To upload objects that takes more than 3 minutes of time in uploading, use the export-async API.

The endpoint has a single optional header, *passphrase*. If you provide the passphrase, the export document will be encrypted using it.

Example CURL command using the optional passphrase header:

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'Authorization: 21c45f0e-82f4-4b04-9072-b49072986231'
--header 'passphrase: my example passphrase'
-d '[
{
"objectIdentifier": {"id": 1},
"objectType": "MASKING_JOB",
"revisionHash": "asdfjkl12jijfdsaklfj21ojasdk"
}
]'
'http://masking-engine.com/masking/api/export'
```

POST /export-async

This endpoint does exactly the same thing as /export, but the execution is done asynchronously. The response returns an async task in the form of this:

```
{
"asyncTaskId": 66,
"operation": "EXPORT",
"reference": "EXPORT-ZXhwb3J0X2RvY3VtZW50XzJjcm1EV09yLmpzb24=",
"status": "RUNNING",
"startTime": "2018-04-13T17:49:55.354+0000",
"cancellable": false
}
```

Example CURL command:

```
curl -s -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'Authorization: 21c45f0e-82f4-4b04-9072-b49072986231'
-d "[
{
"objectIdentifier": {"id": 1},
"objectType": "MASKING_JOB",
"revisionHash": "asdfjkl12jijfdsaklfj21ojasdk"
}
]"
"http://masking-engine.com/masking/api/export-async"
```

The *reference* is used to retrieve the export document of completed async export tasks from the /file-downloads endpoint. The downloaded file from this reference should look exactly the same as the response from /export.

Example CURL command:

```
curl -s -X GET
--header 'Accept: application/octet-stream'
--header 'Authorization: 21c45f0e-82f4-4b04-9072-b49072986231'
-o "<OUTPUT_FILE_PATH>" "http://masking-engine.com/masking/api/file-downloads/EXPORT-
ZXhwb3J0X2RvY3VtZW50XzJjcm1EV09yLmpzb24="
```

Error handling

If an error occurs while exporting one or more elements in the export document, the entire export will abort.

POST /import

```
POST /import?force_overwrite=<true|false>[&environment_id=<id>][&source_environment_id=<id>]
```

This endpoint allows you to import a document exported from another engine. The response returns a list of objects that were imported and whether the import was successful.

The endpoint has one required parameter, *force_overwrite*, two optional parameters *environment_id* and *source_environment_id*, and an optional HTTP header, *passphrase*, which if provided, will cause the engine to attempt to decrypt the document using the specified passphrase. The required *force_overwrite* parameter dictates how to deal with conflicting objects. *environment_id* is necessary for all non-global objects that need to belong in an environment. *source_environment_id* is used for On-The-Fly masking jobs.

The endpoint has a single optional header, passphrase. If you provide the passphrase, the import document will be decrypted using it.

Example CURL command using the optional passphrase header:

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'Authorization: 21c45f0e-82f4-4b04-9072-b49072986231'
--header 'passphrase: my example passphrase'
-d '{
"exportResponseMetadata": {
"exportHost": "masking-engine.com",
"exportDate": "Mon Aug 13 16:29:30 UTC 2018",
"exportedObjectList": [
{
"objectIdentifier": {
"algorithmName": "lookup_alg"
},
"objectType": "LOOKUP",
"revisionHash": "cf84d82c21f0e9d4105d37ae7979c0848486d861"
},
{
"objectIdentifier": {
"keyId": "global"
},
"objectType": "KEY",
"revisionHash": "1d8e9bc552d3ca1dcd218f9e197ea3955ccc29be"
}
]
},
"blob": "<OMITTED>",
"signature": "<OMITTED>", \
"publicKey": "<OMITTED>" \
}'
'http://masking-engine.com/masking/api/import?force_overwrite=true'
```

POST /import-async

```
POST /import-async?force_overwrite=<true|false>[&environment_id=<id>][&source_environment_id=<id>]
```

This endpoint does exactly the same thing as /import, but the execution is done asynchronously and the body is taken in as a file. The response returns an async task in the form of this:

```
{
"asyncTaskId": 67,
"operation": "IMPORT",
"reference": "IMPORT-ZXhwb3J0X2RvY3VtZW50XzJjm1EV09yLmpzb24=",
"status": "RUNNING",
"startTime": "2018-04-13T17:49:55.354+0000",
"cancellable": false
}
```

The *reference* is used to retrieve the import status of completed async import tasks from the /file-downloads endpoint. The downloaded file from this reference should look exactly the same as the response from /import.

Example CURL command:

```
curl -s -X POST
--header 'Content-Type: multipart/form-data'
--header 'Accept: application/json'
--header 'Authorization: 21c45f0e-82f4-4b04-9072-b49072986231'
-F "file=@<DOWNLOADED_FILE_PATH>"
"http://masking-engine.com/masking/api/import-async?force_overwrite=true"
```

Key Management

One important piece of data used by many masking algorithms is the key, which determines the masked outcome of some value. Changing the key changes the output of these algorithms. For example, if the FIRST NAME algorithm masks "Michelle" to "Rachael," changing the key might cause it to mask "Michelle" to "Ben". There are two types of keys that the algorithms can depend on: either 1) global key or 2) individual key.

Global key

The following algorithm types depend on the global key for consistent masked results:

Custom Algorithm* (MAPPLET)

> ✏️ **Note**
>
> A Custom Algorithm does not depend on the global key by nature. However, most mapplets currently used are implemented to use the global key.

A user with Administrator privileges can change the key by clicking the **Generate New Key** button in the **Admin** tab.

> 🔥 **Tip**
>
> Other actions are not allowed during the key generation process. Wait for the **Generate New Key** process to complete and a success dialogue to display in the user interface before performing additional actions on the Masking Engine (e.g., running a masking job).
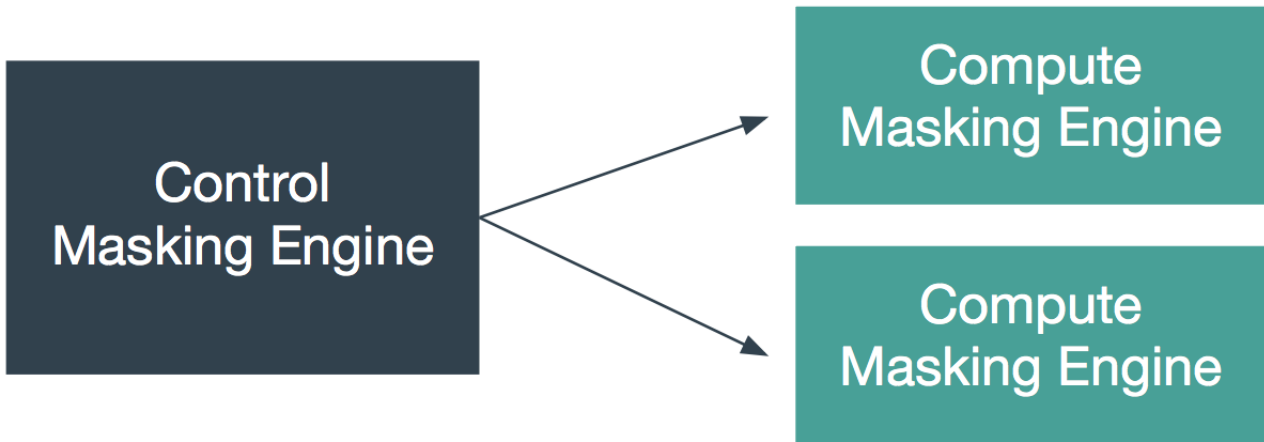
Synchronizing the Global Key between Multiple Engines

In order for Custom Algorithms to behave the same way across several engines, all of those engines must have the same global key. Changing an engine's global key alters the behavior of all of the algorithms that depend on the global key.

You may want to change the key from time to time as a security management practice. If so, change it on all of the engines at the same time. That is, generate a new key on one engine, export that key, and import it to all of the other engines in the deployment.

Keys can be imported and exported independently of algorithms. To export the key from an engine, login to the engine through the login endpoint and then call export with the body shown below. Like all objects, you can encrypt the payload by supplying a passphrase header.

```
[{
"objectIdentifier": {
"keyId": "global"},
"objectType": "KEY"
}]
```

The API will return a JSON payload containing an encoded form of the key that you can install on other engines through the import endpoint. Like all exported objects, it is encoded in an opaque blob.

Individual Key

The following algorithm types have their own key that determines the masked results:

BINARYLOOKUP

DATE_SHIFT (only applies to DateShiftDiscrete)

LOOKUP

TOKENIZATION

The keys for each algorithm gets exported and imported with the algorithm itself, not separately. These individually associated keys can be randomized with an endpoint.

```
PUT http://masking-engine-A/masking/api/algorithms/{algorithmName}/randomize-key
```

Algorithm Syncability

The following tables specify which algorithms are syncable between masking engines (in addition to the masking engine key).

> ✏️ **Note**
>
> Only users with masking admin privilege are able to export and import algorithms.

User-defined Algorithms

Type	Syncable	Workaround
Lookup	Yes	NA
Binary Lookup	Yes	NA
Segmented Mapping	Yes	NA
Mapping	No	None
Tokenization	Yes	NA
Minmax	Yes	NA
Cleansing	Yes	NA
Free Text Redaction	Yes	NA
Custom Algorithm/Mapplet	Yes	NA (see Custom Algorithm)
Component	Yes	NA
## Built-In Algorithms		

> ✏️ **Note**
>
> Syncing built-in algorithms do not actually import the files associated with them

but just updates their individual keys if they have them.

While some of the built-in algorithms are not synchronizable, mainly due to them being non-deterministic, we still can support the export of inventories that contain any built-in algorithm. We just do not guarantee consistent masking of those non-synchronizable built-in algorithms between engines.

Algorithm API Name	Algorithm UI Name	Type	Syncable	Workaround
AccNoLookup	ACCOUNT SL	lookup	Yes	NA
AccountTK	ACCOUNT_TK	tokenization	Yes	NA
AddrLine2Lookup	ADDRESS LINE 2 SL	lookup	Yes	NA
AddrLookup	ADDRESS LINE SL	lookup	Yes	NA
BusinessLegalEntityLookup	BUSINESS LEGAL ENTITY SL	lookup	Yes	NA
CommentLookup	COMMENT SL	lookup	Yes	NA
CreditCard	CREDIT CARD	calculated	No	None
DateShiftDiscrete	DATE SHIFT(DISCRETE)	calculated	Yes	NA
DateShiftFixed	DATE SHIFT(FIXED)	calculated	No	Already synchronized
DateShiftVariable	DATE SHIFT(VARIABLE)	calculated	No	None
DrivingLicenseNoLookup	DR LICENSE SL	lookup	Yes	NA
DummyHospitalNameLookup	DUMMY_HOSPITAL_NAME_SL	lookup	Yes	NA
EmailLookup	EMAIL SL	lookup	Yes	NA
FirstNameLookup	FIRST NAME SL	lookup	Yes	NA
FullNMLookup	FULL_NM_SL	lookup	Yes	NA
LastNameLookup	LAST NAME SL	lookup	Yes	NA
LastCommaFirstLookup	LAST_COMMA_FIRST_SL	lookup	Yes	NA
NameTK	NAME_TK	tokenization	Yes	NA

Algorithm API Name	Algorithm UI Name	Type	Syncable	Workaround
NullValueLookup	NULL SL	lookup	Yes	NA
TelephoneNoLookup	PHONE SL	lookup	Yes	NA
RandomValueLookup	RANDOM_VALUE_SL	lookup	Yes	NA
SchoolNameLookup	SCHOOL NAME SL	lookup	Yes	NA
SecureShuffle	SECURE SHUFFLE	calculated	No	None
SsnTK	SSN_TK	tokenization	Yes	NA
USCountiesLookup	US_COUNTIES_SL	lookup	Yes	NA
USCitiesLookup	USCITIES_SL	lookup	Yes	NA
USstatecodesLookup	USSTATE_CODES_SL	lookup	Yes	NA
USstatesLookup	USSTATES_SL	lookup	Yes	NA
WebURLsLookup	WEB_URLS_SL	lookup	Yes	NA
RepeatFirstDigit	ZIP+4	calculated	No	Already synchronized

Custom Algorithms

Custom algorithms (mapplets) are syncable between masking engines if they are self-contained in the mapplet implementation file. Any other dependencies outside the implementation file, including the masking encryption key, will not be exported from one masking engine and imported into another unless you explicitly manage them. You can manage dependencies on the masking engine encryption key by explicitly requesting the export of the encryption key along with the custom algorithm. Other dependencies, such as data on local file systems or databases (including MDS), must be manually copied from one Delphix Masking Engine to another.

Extensible Algorithms

Extensible Algorithms are fully syncable between Masking Engines. There are two types of extensible algorithms:

a) Built-in into the Algorithm Plugin. Those are synced through synchronization of the corresponding plugin.

b) Configurable extensible algorithms. Those might be synced alone. All the dependencies for extensible algorithms (like other extensible algorithms, files, or external file URLs, mount points for NFS mounted files, algorithm plugins) are automatically synced along with the extensible algorithm. For example: if an extensible algorithm is configured for use by another extensible algorithm - the dependent one (or containing it plugin) is automatically synced during the sync of the main one.

User Workflow examples

This page provides some examples of some typical user workflows. More information on exactly how each endpoint works is available on the Sync Endpoints Section.

Syncing all Global Objects

The following steps can be used to sync all global objects from Masking Engine A to Masking Engine B. This will sync all algorithms and domains and should be done prior to syncing jobs or rulesets which might depend on them. For more information on the global object, see the Sync Concepts Section.

Source Masking Engine Steps

1. Login

Login on the source Masking Engine to obtain an Authorization token value.

```
POST https://a.example.com/masking/api/login

HEADER
Content-Type: application/json
Accept: application/json

BODY
{
  "username": "user123",
  "password": "pw123"
}
```

curl example:

```
curl -X POST --cacert /path/to/cert --header 'Content-Type: application/json' --header 'Accept:
application/json' -d '{ "username": "user123", "password": "pw123" }'
'https://a.example.com/masking/api/login'
```

Expected Result:

```
{
  "Authorization": "dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a"
}
```

2. Get the identifier

Call GET /syncable-objects to obtain the GLOBAL_OBJECT's information.

```
GET https://a.example.com/masking/api/syncable-objects?object_type=GLOBAL_OBJECT

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a (value from the /login response)
Accept: application/json
```

curl example:

```
curl -X GET --cacert /path/to/cert --header 'Accept: application/json' --header 'Authorization: dc2cff8b-
e20d-4e28-8b7e-5d7c4aad0e2a' 'https://a.example.com/masking/api/syncable-objects?object_type=GLOBAL_OBJECT'
```

Expected Result:

```
{
  "_pageInfo": {
    "numberOnPage": 1,
    "total": 1
  },
  "responseList": [
    {
      "objectIdentifier": {
        "id": "global"
      },
      "objectType": "GLOBAL_OBJECT",
      "revisionHash": "8d5236bb029c2176aa568b930786b63253e4f9e4"
    }
  ]
}
```

3. Export the object

Call POST /export-async to asynchronously export the GLOBAL_OBJECT and use the passphrase header to encrypt the export.

> ✏️ **Note**
>
> The optional passphrase header cannot be specified using the interactive API Client tool. An example of how to specify this header using a cURL command is shown below.

```
POST https://a.example.com/masking/api/export-async

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Content-Type: application/json
Accept: application/json
passphrase: my example passphrase

BODY
[
  {
    "objectIdentifier": {
      "id": "global"
    },
    "objectType": "GLOBAL_OBJECT"
  }
]
```

```
curl -X POST --cacert /path/to/cert --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a' --header 'passphrase: my
example passphrase' -d '[{"objectIdentifier":{"id":"global"},"objectType":"GLOBAL_OBJECT"}]'
'https://a.example.com/masking/api/export-async'
```

Expected Result:

```
{
  "asyncTaskId": 2,
  "operation": "EXPORT",
  "reference": "EXPORT-ZXhwb3J0X2RvY3VtZW50Xzk0Wjlva3JDLmpzb24=",
  "status": "RUNNING",
  "startTime": "2018-06-15T20:36:35.483+0000",
  "cancellable": false
}
```

4. Download the export document

Use the reference above to download the export document via the /file-download endpoint.

```
GET https://a.example.com/masking/api/file-downloads/EXPORT-ZXhwb3J0X2RvY3VtZW50Xzk0Wjlva3JDLmpzb24=

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Accept: application/octet-stream
```

curl example:

```
curl -X GET --cacert /path/to/cert --header 'Accept: application/json' --header 'Authorization: dc2cff8b-
e20d-4e28-8b7e-5d7c4aad0e2a' 'https://a.example.com/masking/api/file-downloads/EXPORT-
ZXhwb3J0X2RvY3VtZW50Xzk0Wjlva3JDLmpzb24='
```

Expected Result: An export document that will look like this.

```
{
  "exportResponseMetadata": {
    "exportHost": "a.example.com",
    "exportDate": "Fri Jun 15 20:16:20 UTC 2018",
    "requestedObjectList": [
      {
        "objectIdentifier": {
          "id": "global"
        },
        "objectType": "GLOBAL_OBJECT",
        "revisionHash": "579850b1c88baf74cee6bad61d81e2aa3dcc206c"
      }
    ],
    "exportedObjectList": [
      {
        "objectIdentifier": {
          "id": "DRIVING_LC"
        },
        "objectType": "DOMAIN",
        "revisionHash": "9ee90782488d14d369f9595dad7f593c961e785f"
      },
      {
        "objectIdentifier": {
          "algorithmName": "DrivingLicenseNoLookup"
        },
        "objectType": "LOOKUP",
        "revisionHash": "e08ac9bfd4ed9f64d486cb47cdc07deb30ccc20f"
      },
      ...
    ]
  },
  "blob":
"RAAAAAokZmZhNWIxNjktODMwMC00N2FlLWJjZmMtNjVhNDUzYWI3OTBjEhgyMDE4LTA2LTE1VDIwOjE2OjIwLjY2MFogBSgBFwIAAAokZmZhN
  "signature": "MCwCFAWGf/97wb+oYuSQizj8U12n7jpQAhQKGCaOJ4U8XyDAOEhMUWkzZXHrpw==",
  "publicKey":
"MIHxMIGoBgcqhkjOOAQBMIGcAkEA/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN
}
```

5. Cleanup

When the export document is no longer needed, use the /export-async endpoint to cleanup the exported documents.

```
DELETE https://a.example.com/masking/api/export-async

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Accept: application/json
```

curl example:

```
curl -X DELETE --header 'Accept: application/json' --header 'Authorization: dc2cff8b-e20d-4e28-8b7e-
5d7c4aad0e2a' 'https://a.example.com/masking/api/export-async'
```

Expected Result: no content

Destination Masking Engine Steps

1. Login

Login on the destination Masking Engine to obtain an Authorization token value (see example above).

2. Import the object

On Masking Engine B, use the import-async endpoint to import the document downloaded from engine A.

```
POST https://b.example.com/masking/api/import-async?force_overwrite=true

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Content-Type: multipart/form-data
Accept: application/json
passphrase: my example passphrase
```

curl example:

```
curl -X POST --cacert /path/to/cert --header 'Content-Type: multipart/form-data' --header 'Accept:
application/json' --header 'Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a' --header 'passphrase: my
example passphrase' -F file=@export.json 'https://b.example.com/masking/api/import-async?
force_overwrite=true'
```

Expected Result:

```
{
  "asyncTaskId": 1,
  "operation": "IMPORT",
  "reference": "IMPORT-aW1wb3J0X2RvY3VtZW50X2lZQVFKWEFsLmpzb24=",
  "status": "WAITING",
  "cancellable": false
}
```

3. Verify status

On Masking Engine B, call the /file-downloads endpoint using the reference from the returned Async Task response to retrieve the completed import status.

```
GET https://b.example.com/masking/api/file-downloads/IMPORT-aW1wb3J0X2RvY3VtZW50X2lZQVFKWEFsLmpzb24=

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Accept: application/octet-stream
```

curl example:

```
curl -X GET --cacert /path/to/cert --header 'Accept: application/octet-stream' --header 'Authorization:
dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a' 'https://b.example.com/masking/api/file-downloads/IMPORT-
aW1wb3J0X2RvY3VtZW50X2lZQVFKWEFsLmpzb24='
```

Expected Result:

An import status document that reports the success or failure of each object imported.

```
[
  {
    "objectIdentifier": {
      "id": 7
    },
    "importedObjectIdentifier": {
      "id": 7
    },
    "objectType": "PROFILE_EXPRESSION",
    "importStatus": "SUCCESS"
  },
  {
    "objectIdentifier": {
      "id": "CERTIFICATE_NO"
    },
    "importedObjectIdentifier": {
      "id": "CERTIFICATE_NO"
    },
    "objectType": "DOMAIN",
    "importStatus": "SUCCESS"
  },
  ...
]
```

4. Cleanup

Once the status is no longer needed, use the /import-async endpoint to cleanup the exported documents.

```
DELETE https://b.example.com/masking/api/import-async

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Accept: application/json
```

curl example:

```
curl -X DELETE --cacert /path/to/cert --header 'Accept: application/json' --header 'Authorization:
dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a' 'https://b.example.com/masking/api/import-async'
```

Expected Result: no content

Syncing a Masking Job

The following steps provide an example of how to export a Masking Job from Masking Engine A to Masking Engine B using the synchronous endpoints of /export and /import. This presumes that all of the global objects such as algorithms and domains that the masking job relies on have already been synced. This can also be done via the asynchronous endpoint with the same workflow as above.

1. Export the job

Before this step, the /login and /syncable-objects endpoints should have been called to obtain the authorization token and job identifier respectively. Then use the /export endpoint to obtain an export document with the desired MASKING_JOB. In this example, the optional passphrase is used to encrypt the export document.

> ✏️ **Note**
>
> To sync a profile job, swap out the objectType for "PROFILE_JOB" and provide the id of the profile job to sync. Profile jobs are syncable starting in version 5.3.2.0.

```
POST http://a.example.com/masking/api/export

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Content-Type: application/json
Accept: application/json
passphrase: password to encrypt the export document

BODY
[
  {
    "objectIdentifier": {
      "id": 4
    },
    "objectType": "MASKING_JOB"
  }
]
```

curl example:

```
curl -X POST --cacert /path/to/cert --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a' --header 'passphrase: my
example passphrase' -d '[ { "objectIdentifier": { "id": 4 }, "objectType": "MASKING_JOB" } ]'
'https://a.example.com/masking/api/export'
```

Expected Result:

```
{
  "exportResponseMetadata": {
    "exportHost": "a.example.com",
    "exportDate": "Fri Jun 15 20:16:20 UTC 2018",
    "requestedObjectList": [
      {
        "objectIdentifier": {
          "id": 1
        },
        "objectType": "MASKING_JOB",
        "revisionHash": "579850b1c88baf74cee6bad61d81e2aa3dcc206c"
      }
    ],
    "exportedObjectList": [
      {
        "objectIdentifier": {
          "id": 1
        },
        "objectType": "DATABASE_RULESET",
        "revisionHash": "bf63b401129cbc84f90eeb708281e98121f5a829"
      },
      {
        "objectIdentifier": {
          "id": "FIRST_NAME"
        },
        "objectType": "DOMAIN_REFERENCE",
        "revisionHash": "e6a52079843afd2625f20237fd50f56254c7e630"
      },
      {
        "objectIdentifier": {
          "id": 1
        },
        "objectType": "MASKING_JOB",
        "revisionHash": "579850b1c88baf74cee6bad61d81e2aa3dcc206c"
      },
      {
        "objectIdentifier": {
          "id": 1
        },
        "objectType": "DATABASE_CONNECTOR",
        "revisionHash": "6455f39dfa354a54bdf4ef69d6511a6c2bb19db3"
      },
      {
        "objectIdentifier": {
          "algorithmName": "FirstNameLookup"
        },
        "objectType": "ALGORITHM_REFERENCE",
        "revisionHash": "13b0a51a7e3904f52526c442419c54b39033dca3"
      }
    ]
  },
  "blob":
"RAAAAAokZmZhNWIxNjktODMwMC00N2FlLWJjZmMtNjVhNDUzYWI3OTBjEhgyMDE4LTA2LTE1VDIwOjE2OjIwLjY2MFogBSgBFwIAAAokZmZhN
  "signature": "MCwCFAWGf/97wb+oYuSQizj8U12n7jpQAhQKGCaOJ4U8XyDAOEhMUWkzZXHrpw==",
  "publicKey":
"MIHxMIGoBgcqhkjOOAQBMIGcAkEA/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN
}
```

> ✏️ **Note**
>
> The requestedObjectList returns the list of objects you've requested in the export, and the exportedObjectList returns a list of all objects that were exported. This will include both the requested ones and their dependencies.

2. Import the job

On Masking Engine B, import the masking job. You will need to provide an environment for it to import into.

```
POST http://b.example.com/masking/api/import?force_overwrite=false&environment_id=1

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Content-Type: application/json
Accept: application/json
passphrase: password to encrypt the export document

PARAMETER
force_overwrite and environment_id. See the details in the Masking API Call Concepts section for more
details .

BODY
(Whatever gets returned from export)
```

curl example:

```
curl -X POST --cacert /path/to/cert --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a' --header 'passphrase: my
example passphrase' -d @/path/to/export.json 'https://b.example.com/masking/api/import?
force_overwrite=false&environment_id=1'
```

Expected Result:

```
[
  {
    "objectIdentifier": {
      "id": 3033
    },
    "importedObjectIdentifier": {
      "id": 1
    },
    "objectType": "DATABASE_CONNECTOR",
    "importStatus": "SUCCESS"
  },
  {
    "objectIdentifier": {
      "id": 5421
    },
    "importedObjectIdentifier": {
      "id": 1
    },
    "objectType": "DATABASE_RULESET",
    "importStatus": "SUCCESS"
  }
  ...
]
```

Syncing an Environment

Syncing an environment differs from syncing other objects in that we don't sync any of the environment's metadata, only it's dependencies (jobs, connectors and rulesets). You can think of syncing an environment as an easy way to sync a large group of objects in the environment, without having to sync them one at a time. As such, the environment's revisionHash is not important.

1. Export the environment

Before this step, the /login and /syncable-objects endpoints should have been called to obtain the authorization token and environment identifier respectively. Then use the /export endpoint to obtain an export document with the desired ENVIRONMENT. In this example, the optional passphrase is used to encrypt the export document.

```
POST http://a.example.com/masking/api/export

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Content-Type: application/json
Accept: application/json
passphrase: password to encrypt the export document

BODY
[
  {
    "objectIdentifier": {
      "id": 3
    },
    "objectType": "ENVIRONMENT"
  }
]
```

curl example:

```
curl -X POST --cacert /path/to/cert --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a' --header 'passphrase: my
example passphrase' -d '[ { "objectIdentifier": { "id": 3 }, "objectType": "ENVIRONMENT" } ]'
'https://a.example.com/masking/api/export'
```

Expected Result:

```
{
  "exportResponseMetadata": {
    "exportHost": "a.example.com",
    "exportDate": "Tue Apr 21 21:57:32 UTC 2020",
    "requestedObjectList": [
      {
        "objectIdentifier": {
          "id": 3
        },
        "objectType": "ENVIRONMENT",
        "revisionHash": "c2f2f4bd8a043c32d0977cff8f915d64f1aaf518"
      }
    ],
    "exportedObjectList": [
      {
        "objectIdentifier": {
          "id": 4
        },
        "objectType": "DATASET_CONNECTOR",
        "revisionHash": "db7bc78d098f3df47199fc00c2ba83dee5a52a34"
      },
      {
        "objectIdentifier": {
          "id": 3
        },
        "objectType": "ENVIRONMENT",
        "revisionHash": "c2f2f4bd8a043c32d0977cff8f915d64f1aaf518"
      },
      {
        "objectIdentifier": {
          "id": 4
        },
        "objectType": "MASKING_JOB",
        "revisionHash": "2497260ee897303fc317b9268486c5e36663dad0"
      },
      {
        "objectIdentifier": {
          "id": 4
        },
        "objectType": "DATASET_RULESET",
        "revisionHash": "cb864b0f3f208c4ea5273389055d335d8d57028c"
      },
      {
        "objectIdentifier": {
          "id": 1
        },
        "objectType": "DATASET_FORMAT",
        "revisionHash": "0513a494c736d7f8993dee4720f200c0aa3bd749"
      }
    ]
  },
  "blob": "RAAAAAokZDg5Zjg5NWQtYzJjMi00ZjkyLWIxNjEtMTA0NDRjZDk5YWIxEhgyMDI...",
  "signature": "MCwCFF9wqsdqMG/x7q+knwd4LLhwc4h+AhR9YF5rQZyp5YLQf8e7rI39kjkyUQ==",
  "publicKey": "MIHwMIGoBgcqhkjOOAQBMIGcAkEA/KaCzo4Syrom78z3EQ5SbbB4sF7ey8..."
}
```

> ✏️ **Note**
>
> The requestedObjectList returns the list of objects you've requested in the export, and the exportedObjectList returns a list of all objects that were exported. This will include both the requested ones and their dependencies.

2. Create a new environment on the target engine

Since we do not import the environment metadata (such as name or type) we must first create an environment on the target which we wish to import our data into. At this step we would also need to create a source environment if we are importing any On-The-Fly jobs.

> ✏️ **Note**
>
> All source connectors will end up being imported into the since source environment that we specify. If you wish for these to be in separate environments they must then be manually managed after import.

3. Import the environment into the newly created environment

On Masking Engine B, import the environment. You will need to provide an environment for it to import into.

```
POST http://b.example.com/masking/api/import?force_overwrite=false&environment_id=1

HEADER
Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a
Content-Type: application/json
Accept: application/json
passphrase: password to encrypt the export document

PARAMETER
force_overwrite and environment_id. See the details in the Masking API Call Concepts section for more
details .

BODY
(Whatever gets returned from export)
```

curl example:

```
curl -X POST --cacert /path/to/cert --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: dc2cff8b-e20d-4e28-8b7e-5d7c4aad0e2a' --header 'passphrase: my
example passphrase' -d @/path/to/export.json 'https://b.example.com/masking/api/import?
force_overwrite=false&environment_id=1'
```

Expected Result:

```
[
  {
    "objectIdentifier": {
      "id": 4
    },
    "importedObjectIdentifier": {
      "id": 5
    },
    "objectType": "DATASET_CONNECTOR",
    "importStatus": "SUCCESS"
  },
  {
    "objectIdentifier": {
      "id": 3
    },
    "importedObjectIdentifier": {
      "id": 1
    },
    "objectType": "ENVIRONMENT",
    "importStatus": "SUCCESS"
  },
  {
    "objectIdentifier": {
      "id": 1
    },
    "importedObjectIdentifier": {
      "id": 1
    },
    "objectType": "DATASET_FORMAT",
    "importStatus": "SUCCESS"
  },
  {
    "objectIdentifier": {
      "id": 4
    },
    "importedObjectIdentifier": {
      "id": 5
    },
    "objectType": "DATASET_RULESET",
    "importStatus": "SUCCESS"
  },
  {
    "objectIdentifier": {
      "id": 4
    },
    "importedObjectIdentifier": {
      "id": 5
    },
    "objectType": "MASKING_JOB",
    "importStatus": "SUCCESS"
  }
]
```

Change Log

Changes in 6.0

New Syncable Objects

We added the following new syncable objects in 6.0. Refer to the main documentation for more information on what they are, and how to use them.

- 6.0.0.0 Release
  - MOUNT_INFORMATION
- 6.0.1.0 Release
  - JDBC_DRIVER
  - REIDENTIFICATION_JOB
  - TOKENIZATION_JOB
- 6.0.2.0 Release
  - DATASET_CONNECTOR
  - DATASET_FORMAT
  - DATASET_RULESET
  - ENVIRONMENT
- 6.0.3.0 Release
  - ALGORITHM_PLUGIN
  - USER_ALGORITHM

Changes in 5.3

New Syncable Objects

We added the following new syncable objects in 5.3. Refer to the main documentation for more information on what they are, and how to use them.

- 5.3.0.0 Release
  - DATABASE_RULESET
  - DATE_SHIFT
  - DOMAIN
  - FILE_CONNECTOR
  - FILE_FORMAT
  - FILE_RULESET

- GLOBAL_OBJECT
- MASKING_JOB
- 5.3.3.0 Release
  - PROFILE_EXPRESSION
  - PROFILE_JOB
  - PROFILE_SET

We also added the following new syncable algorithms in 5.3.

- 5.3.2.0 Release
  - CLEANSING
  - MIN_MAX
- 5.3.3.0 Release
  - REDACTION

Key per Algorithm

In pre-5.3, a global key for the engine was used by all algorithms that required a seed to determine the outcome of masked values. This included algorithms such as Lookup and Binary Lookup. Thus, in 5.2, exporting a Lookup Algorithm would automatically export the global encryption key as a dependency. In this release, we allow each algorithm to have its own independent key, exported as a part of the algorithm. Refer to the Key Management section for more detail.

Changed Model of Import Status Reporting

In 5.2, the import status looked like this:

```
{
"objectIdentifier": {
"keyId": "global"
},
"objectType": "KEY",
"importStatus": "SUCCESS"
}
```

Starting in 5.3.0, the import status of an object has extended to include the id or name it has imported into to reduce any confusion introduced with IntegerIdentifiers. For more information on the reason for this change, refer to Logic Behind Overwrite of IntegerIdentifier and StringIdentifier. For examples on what it now looks like, refer to the Example User Workflow section.

Changed Granularity of Transactions for Import

Starting in 5.3, an import of however many objects is performed as an atomic execution rather than using per-object atomicity. This means that the execution will either succeed at importing all objects or fail and import none at all. Refer to the Error Handling of Import logic flow diagram for more information.

Filter for /syncable-objects

Now that we have a large list of syncable objects, we have added a new feature for filtering based on the object type. Refer to the Endpoints page and the Example User Workflow section for more information.

Async Endpoints

Exporting a large MASKING_JOB with many dependencies can potentially take a long time. So we have decided to provide a new endpoint that exports and imports the objects asynchronously. Refer to the Endpoints section in the main documentation and the Example User Workflow page for more information.

Delphix Masking APIs

Masking Client

Masking API Client

This section describes the API client available on the Masking Engine.

Introduction

With the release of API v5 on the Masking Engine, Delphix has opened up the possibility of scripting and automation against the Masking Engine. While this is exciting for us internally at Delphix, we are sure that this will be even more exciting for the consumers of the Masking Engine. This document is intended to be a high-level overview of what to expect with API v5 as well as some helpful links to get you started.

REST

API v5 is a RESTful API. REST stands for REpresentational State Transfer. A REST API will allow you to access and manipulate a textual representation of objects and resources using a predefined set of operations to accomplish various tasks.

JSON

API v5 uses JSON (JavaScript Object Notation) to ingest and return representations of the various objects used throughout various operations. JSON is a standard format and, as such, has many tools available to help with creating and parsing the request and response payloads, respectively.

Here are some UNIX tools that can be used to parse JSON - https://stackoverflow.com/questions/1955505/parsing-json-with-unix-tools. That being said, this is only the tip of the iceberg when it comes to JSON parsing and the reader is encouraged to use their method of choice.

API Client

The various operations and objects used to interact with API v5 are defined in a specification document. This allows us to utilize various tooling to ingest that specification to generate documentation and an API Client, which can be used to generate cURL commands for all operations.

To access the API client on your Masking Engine, go to http://myMaskingEngine.myDomain.com/masking/api-client.

To access the API client documentation without an engine, please refer to the static HTML representations here:

- Masking API 5.0.0 Documentation (released in 5.2.0.0 with incubating endpoints, updated in subsequent releases, and finalized in 5.3.9.0)
- Masking API 5.1.0 Documentation (released in 6.0.0.0)
- Masking API 5.1.1 Documentation (released in 6.0.1.0)
- Masking API 5.1.2 Documentation (released in 6.0.2.0)
- Masking API 5.1.3 Documentation (released in 6.0.3.0)

- [Masking API 5.1.4 Documentation](#) (released in 6.0.4.0)

- [Masking API 5.1.5 Documentation](#) (released in 6.0.5.0)

- [Masking API 5.1.6 Documentation](#) (released in 6.0.6.0)

- [Masking API 5.1.7 Documentation](#) (released in 6.0.7.0)

- [Masking API 5.1.8 Documentation](#) (released in 6.0.8.0)

- [Masking API 5.1.9 Documentation](#) (released in 6.0.9.0)

To see how to log into the API client and for some starter recipes, please check out API Cookbook document.

Supported Features

API v5 is in active development but does not currently support all features that are accessible in the GUI. The list of supported features will expand over the course of subsequent releases.

For a full list of supported APIs, the best place to look is the API client on your Masking Engine.

API Calls for Masking Administration

The Delphix Masking Engine supports the following two types of administrative APIs:

- Analytics APIs

  - These APIs are for including Masking performance information in the support bundle and do not need to be used unless that information is requested.

- Application Setting APIs

  - Application Setting APIs allow an administrator to change the Delphix Masking Engine settings. Presently there are five categories of settings: analytics settings, LDAP settings, general settings, mask settings and profile settings. Over time, more settings will be added to give users direct control over the product's various settings. Below are the details of currently supported settings.

Application Settings APIs

General Group Settings

Setting Group	Setting Name	Type	Description	Default Value
general	EnableMonitorRowCount	Boolean	Controls whether a job displays the total number of rows that are being masked. Setting this to false reduces the startup time of all jobs.	true

	PasswordTimeSpan	Integer [0, ∞)	The number of hours a user is locked out for before they can attempt to log in again.	23
	PasswordCount	Integer [0, ∞)	The number of incorrect password attempts before a user is locked out.	3
	AllowPasswordResetRequest	Boolean	When true, users can request a password reset link be sent to the email associated with their account.	true
	PasswordResetLinkDuration	Integer [1, ∞)	Controls how many minutes the password reset link is valid for.	5
	NumSimulJobsAllowed	Integer [1, ∞)	Max number of jobs allowed.	7
	DefaultApiVersion	String	Used to set default API Version. If the version is omitted from the base path of the request's URL, but wishes to be treated using a specific masking API version that is not the latest version, set the DefaultApiVersion application setting. If the DefaultApiVersion is not set and the version is omitted from the URL, the latest version of the API on that engine will be used. Sample API Version format is like v5.1.5 etc.	Blank

⚠️ **Warning**

NumSimulJobsAllowed setting should be set based on engine configuration. It is risky to run many jobs at once in an environment where you have scheduled more jobs than the system has memory for. When the system runs out of memory all jobs will fail.

Algorithm Group Settings

Setting Group	Setting Name	Type	Description	Default Value
algorithm	DefaultNonConformantDataHandling	String {DONT_MASK,	Default algorithm behavior for	DONT_MASK

		FAIL}		Handling of Non-conformant Data patterns.

Database Group Settings

Setting Group	Setting Name	Type	Description	Default Value
database	DB2zDateFormat	String	Default Date String format to use for DB2 zOS if the database is not using one of the pre-defined IBM DB2 zOS Date String formats. Default is ISO Date String format.	yyyy-MM-dd

LDAP Group Settings

Setting Group	Setting Name	Type	Description	Default Value
ldap	Enable	Boolean	Used to enable and disable LDAP authentication	false
	LdapHost	String	Host of LDAP server	10.10.10.31
	LdapPort	Integer [0, ∞)	Port of LDAP server	389
	LdapBasedn	String	Base DN of LDAP server	DC=tbspune,DC=com
	LdapFilter	String	Filter for LDAP authentication	(&(objectClass=person)(sAMAccountName=?))
	MsadDomain	String	MSAD Domain for LDAP authentication	AD
	LdapTlsEnable	Boolean	Enable and disable the use of TLS for LDAP connections.	false

> ⚠️ **Warning**
>
> In the LDAP group, once the "Enable" setting is set to "true", all users logging in will be authenticated via the LDAP server. Local authentication will no longer work. Before setting this to true set all other LDAP settings correctly and create the necessary LDAP users on the masking engine.

Mask Group Settings

Setting Group	Setting Name	Type	Description	Default Value
mask	DatabaseCommitSize	Integer [1, ∞)	Controls how many rows are updated (Batch Update) to the database before the transaction is committed.	10000
	DefaultStreams	Integer [1, ∞)	Default number of streams for a masking job.	1
	DefaultUpdateThreads	Integer [1, ∞)	Default number of database update threads for a masking job.	1
	DefaultMaxMemory	Integer [1024, ∞)	Default maximum memory for masking jobs (in megabytes).	1024
	DefaultMinMemory	Integer [1024, ∞)	Default minimum memory for masking jobs (in megabytes).	1024

Profile Group Settings

Setting Group	Setting Name	Type	Description	Default Value
profile	EnableDataLevelCount	Boolean	When enabled, only profile the number of rows specified by DataLevelRows when running data level profiling jobs. When disabled, profile all rows when running data level profiling jobs.	false
	DataLevelRows	Integer [1, ∞)	The number of rows a data level profiling job samples when profiling a column. This is only used when	100

			EnableDataLevelCount is true.	
DataLevelPercentage	Double (0, ∞)	Percentage of rows that must match the data level regex to consider this column a match, and thus sensitive.	80.0	
IgnoreDatatype	String	Datatypes that a profiling job should ignore. Columns of these types will not be assigned a domain/algorithm pair.	BIT,BOOLEAN,CHAR#1,VARCHAR#1,V NVARCHAR#1,NVARCHAR2#1,BINARY LOB,LONG,BLOB,CLOB,NCLOB,BFILE	
DefaultStreams	Integer [1, ∞)	Default number of streams for a profiling job.	1	
DefaultMaxMemory	Integer [1024, ∞)	Default maximum memory for profiling jobs (in megabytes).	1024	
DefaultMinMemory	Integer [1024, ∞)	Default minimum memory for profiling jobs (in megabytes).	1024	
OptimizationLevel	Integer [0, 9)	Optimization level for the profiling job which is defined as below, 0: No optimizations are performed. 1: JavaScript runs in interpreted mode. 9: Performs the most optimization with faster script execution, but compiles slower. 1-9: All optimizations are performed.	-1	
DefaultMultiphiAlgorithm	String	Default Multiple PHI masking algorithm which will be used when the Multiple	NullValueLookup	

Profiler Expression will
be true for profile job.

Job Group Settings

Setting Group	Setting Name	Type	Description	Default Value
job	JobLoggingLevel	String {Basic, Detailed}	Controls the amount of information being logged from a job's output. Warning: the Detailed setting may log sensitive information when errors occur. Although this information can be very valuable when debugging a problem, it should be used with care.	Basic

API Calls for Creating Algorithms

Configuring Algorithms

This section provides information on configuring algorithms using the API.

Masking Client Algorithm Model

- **algorithmName** (maxLength=201)

  String

  Equivalent to the algorithm name saved by the user through the GUI. For out of the box algorithms, this will be a similar name as that in the GUI, but presented in a more user-friendly format.

- **algorithmType**

  String

  The type of algorithm

  Enum values:

  - BINARY_LOOKUP

  - CLEANSING

  - COMPONENT

  - LOOKUP

  - MAPPLET

  - MAPPING

  - MINMAX

  - REDACTION

  - SEGMENT

  - TOKENIZATION

- **createdBy** (optional; readOnly; maxLength=255)

  String

  The name of the user that created the algorithm

- **description** (optional; maxLength=255)

  String

  The description of the algorithm

- **frameworkId**

  Integer

  The FrameworkId, corresponding to the framework that extensible algorithm is built upon. This field is to be used only for the Extensible Algorithms. format: int32

- **algorithmExtension** (optional)

  Object

  See specific framework for more details

Algorithm Extension for Extensible Algorithms

It uses the generic Object, defined in the base AlgorithmExtension. Depending on the Extensible Algorithm design it currently supports following implementations (or their mix):

- **fileReference(s)** (optional, name is defined by Extensible Algorithm creator)

  > single *fileReference* or *array[FileReference]*
  >
  > A JSON formatted file reference or list of file references. Each file reference may be one of the following four options: - UUID value returned from the endpoint for uploading file to the Masking Engine - NFS mounted file URL - HTTP URL to external web located file - HTTPS URL to external web located file

- **calledExtendedAlgorithm(s)** (optional, name is defined by Extensible Algorithm creator)

  > single *algorithmName* or *array[AlgorithmName]*
  >
  > A JSON formatted name or a list of extensible algorithms names

Binary Lookup

See Binary Lookup for more information about this algorithm framework.

Binary Lookup Algorithm Extension

- **fileReferenceIds** (optional)

  > *array[String]*
  > A list of file reference UUID values returned from the endpoint for uploading files to the Masking Engine.

Character Mapping

See Character Mapping for more information about this algorithm framework.

Creating a Character Mapping Algorithm via API

1. Retrieve the **frameworkId** for the Character Mapping Framework. This information can be retrieved using the
   following endpoint:

   ```
   algorithm    GET /algorithm/frameworks
   ```

   The framework information should look similar to the following:

   ```
   {
       "frameworkId": 8,
       "frameworkName": "Chracter Mapping",
       "frameworkType" : "STRING",
       "plugin" :
       {
           "pluginId" : 7,
           "pluginName" : "dlpx-core"
       }
   }
   ```

2. Create a Character Mapping algorithm instance via the following endpoint:

   ```
   algorithm    POST /algorithms
   ```

   Configure a new algorithm using the JSON formatted input similar to the following:

   ```
   {
       "algorithmName": "Digits and A to F",
       "algorithmType": "COMPONENT",
       "frameworkId": 8,
       "algorithmExtension": {
           "caseSensitive": true,
           "preserveRanges": null,
           "characterGroups": [
               "0123456789", "[a-fA-F]"
           ],
       "minMaskedPositions": 1,
       "preserveLeadingZeros": false
   }
   ```

Character Mapping Algorithm Extension

- **characterGroups** (required, no default)

Array of Strings

A list of String values defining the characters to be masked. Each group must be either: - a Java regex style character group beginning with '[' - a String of the literal characters that comprise the group.

Duplication of characters within or among groups is not permitted.

- **caseSensitive** (default=true)

   Boolean

   Whether the mapping should be case sensitive. When this is false, each group must be composed either: entirely of characters having no case; or of pairwise matching sets of LC and UC characters - example: [a-zA-Z], not [a-bC-D].

- **minMaskedPositions** (default=1, minimum=0)

   Integer

   The minimum number of positions that must be replaced for masking to be considered successful. Non-conformant data handling is triggered whenever fewer positions are masked. Inputs containing only whitespace never trigger non-conformant data handling.

- **preserveRanges** (optional)

   Array of PreserveRange objects

   A list of PreserveRange objects specifying regions of maskable characters to be preserved. Only maskable characters are considered when determining whether a position is preserved. Ranges are specified with position 0 representing the first maskable character.

- **preserveLeadingZeros** (default=false)

   Boolean

   Whether to preserve leading '0' characters. This option may only be used when '0' is a masked character, and may not be used in conjunction with preserveRanges.

Character Mapping PreserveRange extension

- **start** (minimum=0, required, no default)

   Integer

   The starting position of the preserve range, indexed starting with 0.

- **length** (minimum 1, required, no default)

   Integer

   The length of the preserve range.

- **direction** (default="FORWARD")

   String

   Defines the processing direction for this preserve range, with FORWARD starting at the beginning of input, and REVERSE starting at the end. Possible enum values:

   - FORWARD - process left to right

   - REVERSE - process right to left

Data Cleansing

See Data Cleansing for more information about this algorithm framework.

Data Cleansing Algorithm Extension

- **fileReferenceId** (optional)

  String

  The reference UUID value returned from the endpoint for uploading files to the Masking Engine. The file should contain a newline separated list of {value, replacement} pairs separated by the delimiter. No extraneous whitespace should be present.

- **delimiter** (optional; minLength=1; maxLength=50; default="=")

  String

  The delimiter string used to separate {value, replacement} pairs in the uploaded file

Date Replacement

See Date Replacement for more information about this algorithm framework.

Creating a Date Replacement Algorithm via API

1. Retrieve the **frameworkId** for the Date Replacement Framework. This information can be retrieved using the following endpoint:

```
algorithm    GET /algorithm/frameworks
```

The framework information should look similar to the following:

```
{
    "frameworkId": 1,
    "frameworkName": "Date Replacement",
    "frameworkType" : "LOCAL_DATE_TIME",
    "plugin" :
    {
        "pluginId" : 7,
        "pluginName" : "dlpx-core"
    }
}
```

2. Create a Date Replacement algorithm via the following endpoint:

```
algorithm    POST /algorithms
```

Configure a new algorithm using the JSON formatted input similar to the following:

```
{
    "algorithmName": "exampleDateReplacementAlgorithm",
    "algorithmType": "COMPONENT",
    "frameworkId" : 1,
    "algorithmExtension" :
    {
        "minDate": "2020-01-01 00:00:00",
        "maxDate": "2021-01-01 00:00:00",
        "unit": "DAYS"
    }
}
```

Date Replacement Algorithm Extension

- **minDate**

    String

    A date representing the minimum value that an input can be masked to. The range is inclusive of this value.

- **maxDate**

  > *String*
  > A date representing the maximum value that an input can be masked to. The range is inclusive of this value.

- **unit** (default="SECONDS")

  > *String*
  > A unit of time that determines what the output is truncated to. For example, when the unit is set to days, the years, months, and days may change, but the hours, minutes, and seconds will always be 00:00:00. Unit options supported by this framework: days, hours, minutes, and seconds.

Date Shift

See Date Shift for more information about this algorithm framework.

Creating a Date Shift Algorithm via API

1. Retrieve the **frameworkId** for the Date Shift Framework. This information can be retrieved using the following endpoint:

```
algorithm    GET /algorithm/frameworks
```

The framework information should look similar to the following:

```
{
    "frameworkId": 5,
    "frameworkName": "Date Shift",
    "frameworkType" : "LOCAL_DATE_TIME",
    "plugin" :
    {
        "pluginId" : 7,
        "pluginName" : "dlpx-core"
    }
}
```

2. Create a Date Shift algorithm via the following endpoint:

```
algorithm    POST /algorithms
```

Configure a new algorithm using the JSON formatted input similar to the following:

```
{
    "algorithmName": "exampleDateShiftAlgorithm",
    "algorithmType": "COMPONENT",
    "frameworkId" : 5,
    "algorithmExtension" :
    {
        "minRange" : -3,
        "maxRange" : 3,
        "unit": "MINUTES",
        "roll": "false"
    }
}
```

Date Shift Algorithm Extension

- **minRange**

Integer

A number representing the minimum range value from the input that the input can mask to. The range is inclusive of this value and must be an integer value. Negative values represent units of time in the past and positive values represent units of time in the future. Zero may be included in the range or as one of the range values, but the input will not mask to the same value.

- **maxRange**

  Integer

  A number representing the maximum range value from the input that the input can mask to. The range is inclusive of this value and must be an integer value. Negative values represent units of time in the past and positive values represent units of time in the future. Zero may be included in the range or as one of the range values, but the input will not mask to the same value.

- **unit** (default="DAYS")

  String

  A unit of time that determines what the range is expressed in. Only one unit of time can be specified for each algorithm created. Masked values will be returned with the same granularity as the specified unit. For example a range of 1-2 days will not return the same masked values as a range of 24-48 hours as a range of 1-2 days will return a value with the hours, minutes, and seconds intact but a range of 24-48 hours may return a value with a change in hours anywhere from 24 hours to 48 hours. Unit options supported by this framework: years, months, days, hours, minutes, and seconds.

- **roll** (default="false")

  String

  A boolean that represents whether or not the specified time unit should roll which means that units of time larger and smaller than the specified unit will remain the same. When set to false, there is no guarantee that larger units of time remain the same. When set to true, all larger units of time will retain their same values and the specified unit may wrap around to the beginning. For example, a date at the end of March may wrap around to the beginning of March while keeping all larger units of time and smaller units of time intact. Unit options supported by this framework: months, days, hours, minutes, and seconds.

Dependent Date Shift

See Dependent Date Shift for more information about this algorithm framework.

Creating a Dependent Date Shift Algorithm via API

1. Retrieve the **frameworkId** for the Dependent Date Shift Framework. This information can be retrieved using the following endpoint:

```
algorithm   GET /algorithm/frameworks
```

The framework information should look similar to the following:

```
{
    "frameworkId": 3,
    "frameworkName": "Dependent Date Shift",
    "frameworkType" : "GENERIC_DATA_ROW",
    "plugin" :
    {
        "pluginId" : 7,
        "pluginName" : "dlpx-core"
    }
}
```

2. Create a Dependent Date Shift algorithm via the following endpoint:

```
algorithm   POST /algorithms
```

Configure a new algorithm using the JSON formatted input similar to the following:

```
{
    "algorithmName": "DependentDateShiftTest",
    "algorithmType": "COMPONENT",
    "createdBy": "admin",
    "description": "Test of the DependentDateShiftAlgo",
    "frameworkId": 3,
    "pluginId": 7,
    "fields": [
        {
            "fieldId": 1,
            "name": "date1",
            "type": "LOCAL_DATE_TIME"
        },
        {
            "fieldId": 2,
            "name": "date2",
            "type": "LOCAL_DATE_TIME"
        }
    ],
    "algorithmExtension": {
        "roll": false,
        "unit": "DAYS",
        "maxRange": 5,
        "minRange": 3,
        "intervalRange": 2
    }
}
```

Dependent Date Shift Algorithm Extension

- **minRange**

  Integer

  This number represents the smallest number of time units that will be added to date1 when masking. The range is inclusive of this value. Negative values represent units of time in the past and positive values represent units of time in the future. If date1 is not provided, this is applied to date2.

- **maxRange**

  Integer

  This number represents the largest number of time units that will be added to date1 when masking. The range is inclusive of this value. Negative values represent units of time in the past and positive values represent units of time in the future. If date1 is not provided, this is applied to date2.

- **unit** (default="DAYS")

  String

  A unit of time that the range is expressed in. This unit is also used to determine the interval between date1 and date2. Supported units include years, months, days, hours, minutes, and seconds.

- **roll** (default="false")

String

A boolean that represents whether or not the specified time unit should roll which means that units of time larger and smaller than the specified unit will remain the same. When set to false, there is no guarantee that larger units of time remain the same. Option only supported for months, days, hours, minutes, and seconds. This applies when masking date1. If date1 is not provided, this is applied to date2

- **intervalRange**

  Integer

  A number representing the +/- range value to shift the interval inclusive of the range value. A value of 0 will not change the interval between dates. This number may not be less than 0. If the specified unit difference between date1 and date2 is within the bound of the intervalRange, only values will be provided such that the sign of the difference is preserved. For example, if the day difference between date1 and date2 is 2 and the specified intervalRange is 3, only values greater than -2 will be used (i.e.: -1 to 3). Otherwise, the full range of values will be used (i.e.: -3 to 3).

Email

See Email for more information about this algorithm framework.

Creating an Email Algorithm via API

1. Retrieve the **frameworkId** for the Email Framework. This information can be retrieved using the following endpoint:

```
algorithm    GET /algorithm/frameworks
```

The framework information should look similar to the following:

```
{
     "frameworkId": 3,
     "frameworkName": "Email",
     "frameworkType": "STRING",
     "plugin": {
         "pluginId": 7,
         "pluginName": "dlpx-core",
         "pluginAuthor": "Delphix Engineering",
         "pluginType": "EXTENDED_ALGORITHM"
     }
}
```

   a. Lookup files should be provided via File Reference. Files can be uploaded via the following endpoint:

   fileUpload POST /file-uploads

Alternatively, those files might also be provided via HTTP / HTTPS / NFS mount URLs.

2. Create an Email algorithm via the following endpoint:

```
algorithm    POST /algorithms
```

Configure a new algorithm using the JSON formatted input similar to the following:

```
{
    "algorithmName": "ExampleEmailAlgorithm",
    "algorithmType": "COMPONENT",
    "frameworkId": 3,
    "algorithmExtension": {
        "nameAction": "LOOKUP",
        "domainAction": "REPLACEMENT",
        "nameAlgorithm": null,
        "nameLookupFile": {
            "uri": "delphix-file://upload/f_08bb469a2ddc407bb97a31e96ed0a76a/lookup.txt"
        },
        "domainAlgorithm": null,
        "domainReplacementString": "delphix.com"
    }
}
```

Email Algorithm Extension

- **nameAction**

  NameAction

  The type of action to apply to the name portion of the email. Must be one of the following enum values:

  - *UNIQUE* - applies a SHA-256 hash of the entire input then Base32 encodes the hash value

  - *LOOKUP* - applies a secure lookup using the values provided in the lookup list

  - *APPLY_ALGORITHM* - the name portion is replaced by the output of another chained masking algorithm

- **domainAction**

  MaskAction

  The type of action to apply to the name portion of the email. Must be one of the following enum values:

  - *REPLACEMENT* - the domain portion is replaced by a fixed value

  - *APPLY_ALGORITHM* - the domain portion is replaced by the output of another chained masking algorithm

- **nameLookupFile**

  FileReference

  A file reference to a UTF-8 encoded file containing newline separated replacement values for the name portion of the email.

- **nameAlgorithm**

  AlgorithmInstanceReference

  A reference for the algorithm to use when "APPLY_ALGORITHM" is the NameAction type. The algorithm must have maskingType "STRING". The algorithm will never be passed a null or empty value to mask. See AlgorithmInstanceReference Extension below for more information.

- **domainAlgorithm**

  AlgorithmInstanceReference

  A reference for the algorithm to use when "APPLY_ALGORITHM" is the DomainAction type. The algorithm must have maskingType "STRING". The algorithm will never be passed a null or empty value to mask. See AlgorithmInstanceReference Extension below for more information.

- **domainReplacementString**

String

The string to replace the domain portion when "REPLACEMENT" is the DomainAction type.

AlgorithmInstanceReference Extension

- **name**

  String

  The algorithm instance name.

Free Text Redaction

See Free Text Redaction for more information about this algorithm framework.

Free Text Redaction Algorithm Extension

- **denyListRedaction** (optional; default=true)

  Boolean

  Deny list redaction if true, allow list redaction if false.

- **lookupFileReferenceId** (optional)

  String

  The reference UUID value returned from the endpoint for uploading the lookup file to the Masking Engine.

- **lookupRedactionValue** (optional; maxLength=255)

  String

  The value to use to redact items matching entries specified in the lookup file.

- **profileSetId** (optional)

  Integer

  The ID number of the profile set for defining the pattern matching to use for identifying values for redaction.
  format: int32

- **profileSetRedactionValue** (optional; maxLength=255)

  String

  The value to use to redact items matching patterns defined by the profile set.

Full Name

See Full Name for more information about this algorithm framework.

Creating a Full Name Algorithm via API

1. Find the FrameworkId for the Extensible SL Framework. That might be done via the following EndPoint:

```
algorithm    GET /algorithm/frameworks
```

Plugin name is **dlpx-core**, the framework name is **Full Name**.

2. Involved algorithm references might be built using the name of the desired existing extensible String-type algorithm.
For example: "firstNameAlgorithmRef" : { "name" : "dlpx-core:FirstName" }

3. Create an Extensible Name Algorithm via the following EndPoint:

```
algorithm    POST /algorithms
```

Using the JSON formatted input, similar to the following example:

```
{
    "algorithmName": demo-FullName",
    "algorithmType": "COMPONENT",
    "description": "This is a new style FullName algorithm",
    "frameworkId" : 3,
    "algorithmExtension" :
    {
        "firstNameAlgorithmRef" : { "name" : "dlpx-core:FirstName" },
        "lastNameAlgorithmRef" : { "name" : "dlpx-core:LastName" },
        "maxLengthOfMaskedName" : 0,
        "ifSingleWordConsiderAsLastName" : true,
        "lastNameAtTheEnd" : true,
        "lastNameSeparators" : [ "," ],
        "maxNumberFirstNames" : 2
    }
}
```

Fields description:

"algorithmName" - customer created algorithm name

"algorithmType" - should be "COMPONENT" for Extensible Algorithms

"description" - free text

"frameworkId" - the numeric value found in #1 above

"algorithmExtension" - the composite field, containing algorithm instance specific configuration parameters

Name Algorithm Extension

- **firstNameAlgorithmRef** (required)

  AlgorithmReferenceId
  Must be an Algorithm Reference, pointing to an existing extensible algorithm of String type.

- **lastNameAlgorithmRef** (required)

  AlgorithmReferenceId
  Must be an Algorithm Reference, pointing to an existing extensible algorithm of String type.

- **maxLengthOfMaskedName** (optional, default=0)

  Integer
  Should be a non-negative number. The output (masked) value is forcibly trimmed to that length (by the number of characters).

- **ifSingleWordConsiderAsLastName** (optional)

  Boolean
  If true consider single input word as a last name, otherwise as a first name.
  Default: true

- **lastNameAtTheEnd** (optional)

  Boolean
  If true last name to be detected at the end of the input staring, otherwise last name is at the beginning.
  Default: true

- **lastNameSeparators** (optional)

  List [Char]
  List of the last name separators.
  Default: contains single value: comma ','

- **maxNumberFirstNames** (optional, default=2, minimum=1, maximum=4)

  Integer
  Defines the max number of first and middle names to be masked. The rest would be ignored.

Mapping

See Mapping for more information about this algorithm framework.

Mapping Algorithm Extension

- **fileReferenceId** (optional)

  String

  The reference UUID value returned from the endpoint for uploading files to the Masking Engine. The file should contain a newline separated list of mapping values.

- **ignoreCharacters** (optional; minimum=32; maximum=126)

  array[Integer]

  The integer ASCII values of characters to ignore in the column data to map

Mapplet

Mapplet Extension

- **mappletInput** (optional; maxLength=500)

  String
  The name of the input variable for the custom algorithm

- **mappletOutput** (optional; maxLength=500)

  String
  The name of the output variable for the custom algorithm

- **fileReferenceId** (optional; maxLength=36)

  String
  The reference UUID value returned from the endpoint for uploading files to the Masking Engine.

Min Max

See Min Max for more information about this algorithm framework.

Min Max Algorithm Extension

- **minValue** (optional; minimum=0)

  Integer

  The minimum value for a Number range used in conjunction with maxValue. This field cannot be combined with minDate or maxDate. format: int32

- **maxValue** (optional; minimum=1)

  Integer

  The maximum value for a Number range used in conjunction with and must be greater than minValue. This field cannot be combined with minDate or maxDate. format: int32

- **minDate** (optional)

  date

  The minimum value for a Date range used in conjunction with maxDate. The Date must be specified in one of the following formats according to RFC 3339 Section 5.6: "yyyy-MM-dd", "yyyy-MM-dd'T'HH:mm:ss.SSSZ", "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", or "EEE, dd MMM yyyy HH:mm:ss zzz". If a timezone is not specified, the Date will be interpreted as UTC. This field cannot be combined with minValue or maxValue. format: date

- **maxDate** (optional)

  date

  The maximum value for a Date range used in conjunction with and must be greater than minDate. The Date must be specified in one of the following formats according to RFC 3339 Section 5.6: "yyyy-MM-dd", "yyyy-MM-dd'T'HH:mm:ss.SSSZ", "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", or "EEE, dd MMM yyyy HH:mm:ss zzz". If a timezone is not specified, the Date will be interpreted as UTC. This field cannot be combined with minValue or maxValue. format: date

- **outOfRangeDefaultValue** (optional; maxLength=255)

  String

  The default replacement value for any value that is out-of-range.

Name

See Name for more information about this algorithm framework.

Creating a Name Algorithm via API

1. Find the FrameworkId for the Extensible SL Framework. That might be done via the following EndPoint:

```
algorithm   GET /algorithm/frameworks
```

Plugin name is **dlpx-core**, the framework name is **Name**.

2. Involved files (particlesRoRemoveFile, paticlesToPersistFile, lookupFile) should be provided via the File Reference. For example they can be uploaded via the following EndPoint:

```
fileUpload   POST /file-uploads
```

Alternatively, those files might also be provided via HTTP / HTTPS / NFS mount URLs.

3. Create an Extensible Name Algorithm via the following EndPoint:

```
algorithm   POST /algorithms
```

Using the JSON formatted input, similar to the following example:

```
{
    "algorithmName": "NameDemo",
    "algorithmType": "COMPONENT",
    "description": "This is a new style Name algorithm",
    "frameworkId" :10,
    "algorithmExtension" :
    {
        "filterAccent" : true,
        "particlesToRemoveFile" : {"uri":"delphix-
file://upload/f_1cc829ceee324113ab16c4e750dfce12/particlesToRemove.txt"},
        "maxLengthOfMaskedName" :0,
        "lookupFile":{"uri":"delphix-
file://upload/f_85f082535d054ee8a11696a24ed86d65/LN_LOOKUP_100K%20(1).txt"}
    }
}
```

Fields description:

"algorithmName" - customer created algorithm name

"algorithmType" - should be "COMPONENT" for Extensible Algorithms

"description" - free text

"frameworkId" - the numeric value found in #1 above

"algorithmExtension" - the composite field, containing algorithm instance specific configuration parameters

Name Algorithm Extension

- **lookupFile** (required)

  String

  Lookup file may be FileReferenceId in the one of the following four options:

  - UUID value returned from the endpoint for uploading file to the Masking Engine

  - NFS mounted file URL

  - HTTP URL to external web located file

  - HTTPS URL to external web located file

- **particlesToRemoveFile** (optional)

  String

  File listing particles to remove may be FileReferenceId in the one of the following four options:

  - UUID value returned from the endpoint for uploading file to the Masking Engine

  - NFS mounted file URL

  - HTTP URL to external web located file

  - HTTPS URL to external web located file

- **particlesToPreserveFile** (optional)

  String

  File listing particles to preserve may be FileReferenceId in the one of the following four options:

  - UUID value returned from the endpoint for uploading file to the Masking Engine

  - NFS mounted file URL

  - HTTP URL to external web located file

  - HTTPS URL to external web located file

- **inputCaseSensitive** (optional)

  Boolean

  Setting "true" means input value case matter (i.e. "Peter" and "peter" might be masked to different values).
  Setting "false" (default) makes input value case insensitive ("Peter" and "peter" would be masked to the same value).

- **filterAccent** (optional)

  Boolean

  Setting "true" (default) means accented characters doesn't matter (i.e. "Adrián" and "Adrian" might be masked to the same value).
  Setting "false" makes input value accdnt sensetive ("Adrián" and "Adrian" would be masked to the different values).

- **maskedValueCase** (optional)

> *String*
>
> The output (masked) value case enforcing.
>
> Enum values:
>
> - *PRESERVE_LOOKUP_FILE* - use the unmodified replacement value (default).
>
> - *PRESERVE_INPUT* - preserve case of input value. If mixed - use unmodified replacement value.
>
> - *ALL_LOWER* - force the output to lowercase.
>
> - *ALL_UPPER* - force the output to uppercase.

- **maxLengthOfMaskedName** (optional, default=0)

  > *Integer*
  >
  > Should be a non-negative number. The output (masked) value is forcibly trimmed to that length (by the number of characters).

Payment Card

See Payment Card for more information about this algorithm framework.

Creating a Payment Card Algorithm via API

1. Retrieve the **frameworkId** for the Payment Framework. This information can be retrieved using the following endpoint:

```
algorithm   GET /algorithm/frameworks
```

The framework information should look similar to the following:

```
{
    "frameworkId": 4,
    "frameworkName": "Payment Card",
    "frameworkType" : "STRING",
    "plugin" :
    {
        "pluginId" : 7,
        "pluginName" : "dlpx-core"
    }
}
```

2. Create a Payment Card algorithm via the following endpoint:

```
algorithm   POST /algorithms
```

Configure a new algorithm using the JSON formatted input similar to the following:

```
{
    "algorithmName": "examplePaymentCardAlgorithm",
    "algorithmType": "COMPONENT",
    "frameworkId" : 4,
    "algorithmExtension" :
    {
        "minMaskedPositions" : 7,
        "preserve" : 4
    }
}
```

Payment Card Algorithm Extension

- **minMaskedPositions** (default=1, minValue=0, maxValue=32)

Integer

A value that represents the minimum number of positions that must be replaced for masking to be considered successful. A non-conformant data error is thrown when fewer positions are masked. The minimum value for this field is 0 and the default value is 1. The maximum value is 32.

- **preserve** (default=0, minValue=0, maxValue=32)

  Integer

  A value that represents the number of maskable characters to preserve at the beginning of the input. Only maskable characters are considered when determining whether a position is preserved. The minimum value for this field is 0 and the default value is 0. The maximum value is 32.

Regex Decompose

See Regex Decompose for more information about this algorithm framework.

Creating a Regex Decompose Algorithm via API

1. Retrieve the **frameworkId** for the Regex Decompose Framework. This information can be retrieved using the following endpoint:

```
algorithm   GET /algorithm/frameworks
```

The framework information should look similar to the following:

```
{
    "frameworkId": 5,
    "frameworkName": "Regex Decompose",
    "frameworkType": "STRING",
    "plugin": {
        "pluginId": 7,
        "pluginName": "dlpx-core",
        "pluginAuthor": "Delphix Engineering",
        "pluginType": "EXTENDED_ALGORITHM"
    }
}
```

2. Create a Regex Decompose algorithm via the following endpoint:

```
algorithm   POST /algorithms
```

Configure a new algorithm using the JSON formatted input similar to the following:

```
{
    "algorithmName": "ExampleRegexDecomposeAlgorithm",
    "algorithmType": "COMPONENT",
    "frameworkId": 5,
    "algorithmExtension": {
        "trimInput": true,
        "requireMask": false,
        "maskPatterns": [
            {
                "regex": "([0-9]+)-([a-z]+)",
                "actions": [
                    {
                        "type": "REDACT",
                        "algorithm": null,
                        "redactString": "asdf",
                        "redactCharacter": null
                    },
                    {
                        "type": "REDACT",
                        "algorithm": null,
                        "redactString": null,
                        "redactCharacter": "x"
                    }
                ]
            }
        ],
        "fallbackAction": null,
        "maxInputLength": 65536
    }
}
```

Regex Decompose Algorithm Extension

- **maskPatterns**

  Array of *MaskPattern* objects
  Defines the mask pattern(s) of the algorithm. See Regex Decompose MaskPattern Extension below for more information.

- **fallbackAction**

  MaskAction
  The action that should be applied to the entire input if none of the defined regular expressions match. If no pattern matches and no fallbackAction is set, non-conformant data handling will be triggered. See Regex Decompose MaskAction Extension below for more information.

- **requireMask** (default="true")

  String
  A boolean that represents whether the input must be masked. When this is true, patterns are matched until one changes the input. If no pattern can change the input and no fallbackAction is set, non-conformant data handling will be triggered for this value. If false, the first matching pattern will apply regardless of whether it changes the input. Any difference in value from the input is considered successful masking.

- **trimInput** (default="true")

String

A boolean that represents whether to trim whitespace from the beginning and end of the input prior to processing. The same leading and trailing whitespace will be reintroduced into the masked value. This option is provided to simplify the regular expressions that can be used in maskPatterns, as they no longer must account for and preserve leading and trailing whitespace.

- **maxInputLength** (default=65536, minValue=1)

  Integer

  A value that represents the maximum character length of input the algorithm will attempt to process. If the input length exceeds this value, non-conformant data handling will be triggered for this value.

Regex Decompose MaskPattern Extension

- **regex**

  String

  A Java 8 style regular expression used to match the masking input.

- **actions**

  Array of *MaskAction* objects
  Defines the action(s) to be applied to the match or capturing group(s) when the regular expression matches. See Regex Decompose MaskAction Extension below for more information.

Regex Decompose MaskAction Extension

- **type**

  String

  The type of action to the input that matches the regex. Must be one of the following enum values:
  - *PRESERVE* - the value or capturing group is not masked and remains unchanged
  - *TRUNCATE* - the value or capturing group is replaced with ""
  - *REDACT* - the value or capturing group is replaced by a value or repeated character
  - *APPLY_ALGORITHM* - the value or capturing group is replaced by the output of another chained masking algorithm

- **algorithm**

  AlgorithmInstanceReference

  A reference for the algorithm to use when "APPLY_ALGORITHM" is the MaskAction type. The algorithm must have maskingType "STRING". The algorithm will never be passed a null or empty value to mask. See AlgorithmInstanceReference Extension below for more information.

- **redactCharacter**

  String

  The character to use to replace the input when "REDACT" is the MaskAction type. Each character in the portion of input matched is replaced with this character. Length of the matched input is preserved. Only one of redactCharacter or redactString may be specified for a given MaskAction.

- **redactString**

  String

  The string to use to replace the input when "REDACT" is the MaskAction type. The entire matched portion is replaced with this string. Use of this option will cause the length of the value to change during masking unless the matched portion of input happens to have the same length of the redactString. Only one of redactCharacter or redactString may be specified for a given MaskAction.

AlgorithmInstanceReference Extension

- **name**

  String

  The algorithm instance name.

Secure Lookup

See Secure Lookup for more information about this algorithm framework.

Creating a Secure Lookup Algorithm via API

1. Find the FrameworkId for the Extensible SL Framework. That might be done via the following EndPoint:

```
algorithm    GET /algorithm/frameworks
```

   Plugin name is **dlpx-core**, the framework name is **Secure Lookup**.

2. Upload Lookup File via the following EndPoint:

```
fileUpload    POST /file-uploads
```

   Alternatively, the Lookup File might also be provided via HTTP / HTTPS / NFS mount URLs.

3. Create an Extensible SL Algorithm via the following EndPoint:

```
algorithm    POST /algorithms
```

   Using the JSON formatted input, similar to the following example:

```
{
    "algorithmName": "demoExtendedSL",
    "algorithmType": "COMPONENT",
    "frameworkId" : 1,
    "algorithmExtension" :
    {
        "lookupFile" : {
            "uri":"delphix-file://upload/f_7984ee9672b44e309f7ef5940f856e7c/ColorsLF.txt"
        },
        "inputCaseSensitive" : true,
        "maskedValueCase" : "ALL_LOWER"
    }
}
```

   Fields description:

   "algorithmName" - customer created algorithm name

   "algorithmType" - should be "COMPONENT" for Extensible Algorithms

   "description" - free text

   "frameworkId" - the numeric value found in #1 above

   "algorithmExtension" - the composite field, containing algorithm instance specific configuration parameters

Secure Lookup Algorithm Extension

- **lookupFile** (maxLength=255)

  String

  Lookup file may be one of the following four options:

  - UUID value returned from the endpoint for uploading file to the Masking Engine

  - NFS mounted file URL

  - HTTP URL to external web located file

  - HTTPS URL to external web located file

- **inputCaseSensitive** (optional)

  Boolean

  Setting "true" means input value case matter (i.e. "Peter" and "peter" might be masked to different values)

  Setting "false" (default) makes input value case insensitive ("Peter" and "peter" would be masked to the same value)

- **maskedValueCase** (optional)

  String

  The output (masked) value case enforcing.

  Enum values:

  - *PRESERVE_LOOKUP_FILE* - use the unmodified replacement value (default).

  - *PRESERVE_INPUT* - preserve case of input value. If mixed - use unmodified replacement value.

  - *ALL_LOWER* - force the output to lowercase.

  - *ALL_UPPER* - force the output to uppercase.

Legacy Secure Lookup Algorithm Extension

- **fileReferenceId** (optional)

  String

  The reference UUID value returned from the endpoint for uploading files to the Masking Engine.

Segment Mapping

See Segment Mapping for more information about this algorithm framework.

Segment Mapping Algorithm Extension

- **preservedRanges** (optional)

  array[SegmentMappingPreservedRange]
  List of character {offset, length} values specifying ranges of the real value to preserve. Offsets begin at 0

- **ignoreCharacters** (optional)

  array[Integer]
  List of decimal values specifying ASCII characters to ignore (not mask, not count as part of any segment) in the real value. For example, 65 would ignore 'A'

- **segments** (optional; minItems=2; maxItems=36)

  array[SegmentMappingSegment]

Segment Mapping Preserve Range Extension

- **offset** (optional)

  Integer
  The character offset of the range of input to preserve

- **length** (optional)

  Integer
  The character length of the range of input to preserve

Segment Mapping Segment Extension

- **length** (optional; minimum=1; maximum=4)

  Integer
  The length of the segment in digits. This must be 1 for alpha-numeric segments

- **minInt** (optional; minimum=0; maximum=9999)

  Integer
  The minimum value of the integer output range of the mapping function

- **maxInt** (optional; minimum=0; maximum=9999)

  Integer
  The maximum value of the integer output range of the mapping function

- **minChar** (optional; minLength=1; maxLength=1)

> *String*
> The minimum value of the character output range of the mapping function

- **maxChar** (optional; minLength=1; maxLength=1)

  > *String*
  > The maximum value of the character output range of the mapping function

- **explicitRange** (optional)

  > *String*
  > Explicitly specify the output range. Format depends on segment type and size

- **minRealInt** (optional; minimum=0; maximum=9999)

  > *Integer*
  > The minimum value of the integer range specifying which real values will be masked

- **maxRealInt** (optional; minimum=0; maximum=9999)

  > *Integer*
  > The maximum value of the integer range specifying which real values will be masked

- **minRealChar** (optional; minLength=1; maxLength=1)

  > *String*
  > The minimum value of the character range specifying which real values will be masked

- **maxRealChar** (optional; minLength=1; maxLength=1)

  > *String*
  > The maximum value of the character range specifying which real values will be masked

- **explicitRealRange** (optional)

  > *String*
  > Explicitly specify the range of input values that should be masked. Format depends on segment type and size

API Calls for Managing Extended Connectors

Introduction

This section details how to manage extended database connectors, including how to manage driver support tasks on a masking job.

1. Installing A Driver Support Plugin

2. Installing A JDBC Driver

3. Creating An Extended Database Connector

4. Managing Masking Job Driver Support Tasks

> ✏️ **Note**
>
> Installing a JDBC driver with a driver support is only possible via the web API.

Install Driver Support jar on Masking Engine

1. Select `POST /file-uploads`

2. Click "Choose File" and select desired driver support jar

The response will look similar to the following with a return status of 200:

```
{
    "fileReferenceId": "delphix-file://upload/f_xxxx/sampleDriverSupport.jar"
}
```

Create Driver Support Plugin

1. Select `POST /plugins`

2. **fileReferenceId**: delphix-file://upload/f_xxxx/sampleDriverSupport.jar

3. **pluginName**: whatever desired name

4. **pluginType**: DRIVER_SUPPORT

The response will look similar to the following with a return status of 200:

```
{
    "pluginId": 9,
    "pluginName": "Sample Plugin",
    "pluginAuthor": "Sample Plugin Author",
    "pluginType": "DRIVER_SUPPORT",
    "originalFileName": "driverSupport.jar",
    "originalFileChecksum": "f8398c0768ecf7709c6992b3f048f9da8be640285b3ccc968973949ca3cceb02",
    "installDate": "2021-04-21T15:29:01.982+00:00",
    "installUser": 5,
    "builtIn": false,
    "pluginVersion": "1.5.0",
    "pluginObjects": [
        {
            "objectIdentifier": "1",
            "objectName": "Disable Constraints",
            "objectType": "DRIVER_SUPPORT_TASK"
        },
        {
            "objectIdentifier": "2",
            "objectName": "Disable Triggers",
            "objectType": "DRIVER_SUPPORT_TASK"
        },
        {
            "objectIdentifier": "3",
            "objectName": "Drop Indexes",
            "objectType": "DRIVER_SUPPORT_TASK"
        }
    ]
}
```

> **ℹ️ Info**
>
> The `objectIdentifier` field refers to the ID of the task. Specifying the ID of the tasks is required to enable tasks on a masking job. The order in which the tasks are returned from the API is the order in which the tasks will be executed; the `objectIdentifier` (task ID) has no bearing on the task execution order.

Create JDBC Driver that Uses Driver Support Plugin

1. Select `POST /jdbc-drivers`

2. Form the request body as follows:

```
{
    "driverName": "HANA driver",
    "driverClassName": "com.sap.db.jdbc.Driver",
    "fileReferenceId": "delphix-file://upload/f_xxxx/sampleJdbcDriver.zip",
    "driverSupportId": 9
}
```

The response will look similar to the following with a return status of 200:

```
{
    "jdbcDriverId": 8,
    "driverName": "HANA driver",
    "driverClassName": "com.sap.db.jdbc.Driver",
    "version": "2.4",
    "uploadedBy": "admin",
    "uploadDate": "2021-04-27T20:34:47.748+00:00",
    "checksum": "a5b7cf1323b71398e68fd583cd4f40ef8a5f4212ae94b63e95c904ed226d4c7b",
    "builtIn": false,
    "loggerInstalled": true,
    "driverSupportId": 9
}
```

> **⚠️ Warning**
>
> If the referenced driver support plugin is being used by existing masking jobs that have tasks enabled, extra validation is performed. In the case of updating a driver support plugin or updating a JDBC driver to use a different driver support, the driver support plugin must implement all enabled tasks on any existing masking job. If the other driver support does not implement all enabled tasks, the update will fail. In the case of deleting a driver support plugin, the delete will fail if the driver support plugin is being used by any existing masking jobs that have tasks enabled.

Install JDBC Driver zip on Masking Engine

1. Select `POST /file-uploads`

2. Click "Choose File" and select desired JDBC driver zip

The response will look similar to the following with a return status of 200:

```
{
    "fileReferenceId": "delphix-file://upload/f_xxxx/sampleJdbcDriver.zip"
}
```

> ✏️ **Note**
>
> Note that you can also install a JDBC driver via the UI.

Create JDBC Driver without Driver Support

1. Select `POST /jdbc-drivers`

2. Form the request body as follows:

```
{
    "driverName": "HANA driver",
    "driverClassName": "com.sap.db.jdbc.Driver",
    "fileReferenceId": "delphix-file://upload/f_xxxx/sampleJdbcDriver.zip",
}
```

The response will look similar to the following with a return status of 200:

```
{
    "jdbcDriverId": 8,
    "driverName": "HANA driver",
    "driverClassName": "com.sap.db.jdbc.Driver",
    "version": "2.4",
    "uploadedBy": "admin",
    "uploadDate": "2021-04-27T20:34:47.748+00:00",
    "checksum": "a5b7cf1323b71398e68fd583cd4f40ef8a5f4212ae94b63e95c904ed226d4c7b",
    "builtIn": false,
    "loggerInstalled": true,
}
```

Create JDBC Driver with Driver Support

Follow the same process to create a JDBC driver without a driver support, only add `driverSupportId` to the request body to indicate the ID of the driver support to associate with the driver.

> ⚠️ **Warning**

If the JDBC driver's referenced driver support plugin tasks are enabled on any existing masking job, validation on update is done in order to prevent changing the driver support plugin to another one unless it implements all enabled tasks. If the other driver support does not implement all enabled tasks, the update will fail.

Create An Extended Database Connector

> **ℹ️ Info**
>
> This assumes an application and environment already exists, to which you can add this extended connector.

1. Select `POST /database-connectors`

2. Format response body as follows:

```
{
    "connectorName": "hana db",
    "databaseType": "EXTENDED",
    "environmentId": 1,
    "jdbc": "JDBC_SERVER_URL",
    "username": "USERNAME",
    "password": "PASSWORD",
    "kerberosAuth": false,
    "jdbcDriverId": 7,
    "enableLogger": false
}
```

The response will look similar to the following with a return status of 200:

```
{
    "databaseConnectorId": 1,
    "connectorName": "hana db",
    "databaseType": "EXTENDED",
    "environmentId": 1,
    "jdbc": "JDBC_SERVER_URL",
    "username": "USERNAME",
    "kerberosAuth": false,
    "jdbcDriverId": 7,
    "enableLogger": false
}
```

Managing Masking Job Driver Support Tasks

Enabling tasks is possible via `POST /jobs` and `PUT /jobs/{job_id}`. Disabling tasks is possible via `PUT /jobs/{job_id}`.

Create Masking Job That Enables Tasks

> **ℹ Info**
>
> This assumes a ruleset using the desired extended connector already exists. The following example demonstrates the creation of an in-place masking job on an extended connector. This also assumes you know the ID of the task that you want to enable and have execute as part of a given masking job.

1. Create a masking job using the ruleset you created earlier that targets the extended connector

2. Format the request body as follows:

```
{
    "jobName": "Hana IP job",
    "rulesetId": 1,
    "jobDescription": "Job description",
    "onTheFlyMasking": false,
    "enabledTasks": [
        {
            "taskId": 1
        },
        {
            "taskId": 3
        }
    ]
}
```

> **ℹ Info**
>
> The request body is almost exactly the same as the example API calls for creating and running masking jobs; the difference is that to enable tasks to execute as part of a masking job on an extended connector, you need to ensure the ruleset points to an extended connector that is using a JDBC driver with a driver support and include the property `enabledTasks` in your request.

The response will look similar to the following with a return status of 200:

```json
{
    "maskingJobId": 1,
    "jobName": "Hana IP job",
    "rulesetId": 1,
    "rulesetType": "table",
    "createdBy": "admin",
    "createdTime": "2021-04-27T21:29:46.043+00:00",
    "feedbackSize": 50000,
    "jobDescription": "Job description",
    "maxMemory": 1024,
    "minMemory": 1024,
    "multiTenant": false,
    "numInputStreams": 1,
    "onTheFlyMasking": false,
    "databaseMaskingOptions": {
        "batchUpdate": true,
        "commitSize": 10000,
        "disableConstraints": false,
        "dropIndexes": false,
        "disableTriggers": false,
        "numOutputThreadsPerStream": 1,
        "truncateTables": false
    },
    "failImmediately": false,
    "enabledTasks": [
        {
            "taskId": 1
        },
        {
            "taskId": 3
        }
    ],
    "streamRowLimit": 20000
}
```

Disable Tasks

Using the above request body as an example, to disable the formerly enabled task with an ID of 1, the request body to `PUT /masking-jobs/1` should exclude the object in `enabledTasks` with `"taskId": 1`. That is, the request body should be:

```json
{
    "jobName": "Hana IP job",
    "rulesetId": 1,
    "jobDescription": "Job description",
    "onTheFlyMasking": false,
    "enabledTasks": [
        {
            "taskId": 3
        }
    ]
}
```

The job will now only have the task with an ID of 3 enabled. The response will look similar to the following with a return status of 200:

```
{
    "maskingJobId": 1,
    "jobName": "Hana IP job",
    "rulesetId": 1,
    "rulesetType": "table",
    "createdBy": "admin",
    "createdTime": "2021-04-27T21:29:46.043+00:00",
    "feedbackSize": 50000,
    "jobDescription": "Job description",
    "maxMemory": 1024,
    "minMemory": 1024,
    "multiTenant": false,
    "numInputStreams": 1,
    "onTheFlyMasking": false,
    "databaseMaskingOptions": {
        "batchUpdate": true,
        "commitSize": 10000,
        "disableConstraints": false,
        "dropIndexes": false,
        "disableTriggers": false,
        "numOutputThreadsPerStream": 1,
        "truncateTables": false
    },
    "failImmediately": false,
    "enabledTasks": [
        {
            "taskId": 3
        }
    ],
    "streamRowLimit": 20000
}
```

API Calls for Creating an Inventory

Below are examples of requests you might enter and responses you might receive from the Masking API client. For commands specific to your masking engine, work with your interactive client at http://**<myMaskingEngine&gt**/masking/api-client/

> ⚠️ **Warning**
>
> HTTPS (SSL/TLS) is recommended, but for explanatory purposes these examples use insecure HTTP

> ℹ️ **Info**
>
> In all code examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

Fetch Table Names from Database Connector

Object references you will need:

- The ID of the database connector to fetch tables for

> ✏️ **Note**
>
> This database connector ID (1, in this example) is included in the PATH for this operation, NOT the payload.

REQUEST

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0'
'http://<myMaskingEngine>/masking/api/database-connectors/1/fetch'
```

RESPONSE

```
[ "ALL_COLUMNS", "DBVERIFICATION_TABLE"]
```

More info

http://<myMaskingEngine>/masking/api-client/#!/databaseConnector/fetchTableMetadata

Example

See how to use this in the context of a script here.

Create Table Metadata

Object references you will need:

- The name of the table to create the metadata for
- The ruleset ID

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization:
7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0' -d '{ "tableName": "ALL_COLUMNS",
"rulesetId": 2 }'
'http://<myMaskingEngine>/masking/api/table-metadata'
```

RESPONSE

```
{ "tableMetadataId": 2, "tableName": "ALL_COLUMNS", "rulesetId": 2
}
```

More info

http://<myMaskingEngine>/masking/api-client/#!/tableMetadata/createTableMetadata

Example

See how to use this in the context of a script here.

Get All Column Metadata Belonging to Table Metadata

Object references you will need:

- The table metadata ID to get the columns for

> 🔥 **Tip**
>
> This table metadata ID (2, in this example) is included in the QUERY STRING for this operation, NOT the payload.

REQUEST

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0'
'http://<myMaskingEngine>/masking/api/column-metadata?table_metadata_id=2'
```

RESPONSE

```
[ { "columnMetadataId": 12, "columnName": "schoolnme",
"tableMetadataId": 2, "columnLength": 50, "isMasked": false,
"isPrimaryKey": false, "isIndex": false, "isForeignKey": false }, … ]
```

Note that the above response has been truncated due to its length for the purposes of this documentation.

More info

http://<myMaskingEngine>/masking/api-client/#!/columnMetadata/getAllColumnMetadata

Example

See how to use this in the context of a script here.

Update Column Metadata with Algorithm Assignment

Object references you will need:

- Column metadata ID for the column you wish to update

> 🔥 **Tip**
>
> This column metadata ID (20, in this example) is included in the PATH for this operation, NOT the payload.

- Since the names can vary in the API and UI, you should use the names obtained through the API (these may not align with the UI).
- Algorithm name
- Domain name

REQUEST

```
curl -X PUT --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization:
7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0' -d '{ "algorithmName":
"AddrLine2Lookup", "domainName": "ADDRESS_LINE2", "isProfilerWritable": false }'
'http://<myMaskingEngine>/masking/api/column-metadata/20'
```

RESPONSE

```
{ "columnMetadataId": 20, "columnName": "l2_address",
"tableMetadataId": 2, "algorithmName": "AddrLine2Lookup", "domainName":
"ADDRESS_LINE2", "columnLength": 512, "isMasked": true, "isProfilerWritable": false, "isPrimaryKey":
false, "isIndex": false, "isForeignKey": false
}
```

More info

http://<myMaskingEngine>/masking/api-client/#!/columnMetadata/updateColumnMetadata

Example

See how to use this in the context of a script here.

API Calls for Creating and Running Masking Jobs

Below are examples of requests you might enter and responses you might receive from the Masking API client. For commands specific to your masking engine, work with your interactive client at http://**<myMaskingEngine>**/masking/api-client/

> ✏️ **Note**
>
> In all code examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

> ⚠️ **Warning**
>
> HTTPS (SSL/TLS) is recommended, but for explanatory purposes these examples use insecure HTTP.

Creating a Masking Job

Object references you will need:

- The ID of the ruleset for which you wish to create the masking job

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization:
e23bad24-8760-4091-a131-34f235d9b2d6' -d '{ "jobName":
"some_masking_job", "rulesetId": 7, "jobDescription": "This example
illustrates a MaskingJob with just a handful of the possible fields set.
It is meant to exemplify a simple JSON body that can be passed to the
endpoint to create a MaskingJob.", "feedbackSize": 100000,
"onTheFlyMasking": false }'
'http://<myMaskingEngine>/masking/api/masking-jobs'
```

RESPONSE

```
{ "jobId": 1, "jobName": "some_masking_job", "rulesetId": 7,
"createdBy": "Axistech", "createdTime": "2017-07-04T00:31:00.952+0000",
"environmentId": 2, "feedbackSize": 100000, "jobDescription": "This
example illustrates a MaskingJob with just a handful of the possible
fields set. It is meant to exemplify a simple JSON body that can be
passed to the endpoint to create a MaskingJob.", "maxMemory": 1024,
"minMemory": 1024, "multiTenant": false, "numInputStreams": 1,
"onTheFlyMasking": false }
```

> ✏️ **Note**
>
> The response includes the ID of the newly created job ("jobId").

More info

http://<myMaskingEngine>/masking/api-client/#!/maskingJob/createMaskingJob

Running a Masking Job

Create a new execution of a masking job.

Object references you will need:

- The ID of the job you want to run

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization:
e23bad24-8760-4091-a131-34f235d9b2d6' -d '{ "jobId": 1 }'
'http://<myMaskingEngine>/masking/api/executions'
```

RESPONSE

```
{ "executionId": 1, "jobId": 1, "status": "RUNNING"
}
```

More info

http://<myMaskingEngine>/masking/api-client/#!/execution/createExecution

Checking the Status of a Masking Job

Object references you will need:

- The ID of the execution you want to check (IN THE PATH)

> ✏ **Note**
>
> This execution id (1, in this example) is included in the PATH for this operation, NOT the payload.

REQUEST

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
8935f7f7-6de6-40ba-80d8-d8956b71248b'
'http://<myMaskingEngine>/masking/api/executions/1'
```

RESPONSE

```
{
  "executionId": 1,
  "jobId": 1,
  "status": "SUCCEEDED",
  "rowsMasked": 1000,
  "rowsTotal": 1000,
  "startTime": "2019-02-14T21:51:13.253+0000",
  "endTime": "2019-02-14T21:51:54.956+0000"
}
```

More info

http://<myMaskingEngine>/masking/api-client/#!/execution/getExecutionById

Retrieving Execution Events related to a Masking Job

Object references you will need:

- The ID of the execution you want to check (as a URL parameter).

> ✏️ **Note**
>
> This execution id (1, in this example) is specified as a URL parameter for this operation.
>
> The execution-events endpoint returns execution events for a specified job execution. These execution events report failures or warnings associated with the masking job execution. NOT specifying the execution in the URL parameter will retrieve all execution events for all masking jobs.

REQUEST

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
8935f7f7-6de6-40ba-80d8-d8956b71248b'
'http://<myMaskingEngine>/masking/api/execution-events?execution_id=1&page_number=1'
```

RESPONSE

```json
{
  "_pageInfo": {
    "numberOnPage": 1,
    "total": 1
  },
  "responseList": [
    {
      "executionEventId": 1,
      "executionId": 1,
      "eventType": "UNMASKED_DATA",
      "severity": "WARNING",
      "cause": "PATTERN_MATCH_FAILURE",
      "count": 1000,
      "timeStamp": "2019-02-14T21:51:51.790+0000",
      "executionComponentId": 1,
      "maskedObjectName": "RCHARS64_T1_0",
      "algorithmName": "DateShiftVariable"
    }
  ]
}
```

More info

http://<myMaskingEngine>/masking/api-client/#!/execution-events/getAllExecutionEvents

Retrieving Non-conformant Data Samples associated with an Execution Event

Object references you will need:

- The ID(s) of the execution event(s) you want to check (as one or more URL parameters).

> ✏️ **Note**
>
> This execution event id (1, in this example) is specified as a URL parameter for this operation.
>
> The non-conformant-data-sample endpoint returns non-conformant data samples for a specified job execution event. These non-conformant data samples will report the data patterns that caused the non-conformant data execution event to help identify why data is not getting masked. NOT specifying an execution event in the URL parameter will retrieve all non-conformant data samples events for all masking jobs.

REQUEST

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
8935f7f7-6de6-40ba-80d8-d8956b71248b'
'http://<myMaskingEngine>/masking/api/non-conformant-data-sample?execution_event_id=1&page_number=1'
```

RESPONSE

```
{
  "_pageInfo": {
    "numberOnPage": 7,
    "total": 7
  },
  "responseList": [
    {
      "dataSampleId": 1,
      "executionEventId": 1,
      "dataSample": "LLLLL",
      "count": 200
    },
    {
      "dataSampleId": 2,
      "executionEventId": 1,
      "dataSample": "LLLLLL",
      "count": 400
    },
    {
      "dataSampleId": 3,
      "executionEventId": 1,
      "dataSample": "LLLL",
      "count": 80
    },
    {
      "dataSampleId": 4,
      "executionEventId": 1,
      "dataSample": "LLLLLLL",
      "count": 100
    },
    {
      "dataSampleId": 5,
      "executionEventId": 1,
      "dataSample": "LLLLLLLLLLL",
      "count": 50
    },
    {
      "dataSampleId": 6,
      "executionEventId": 1,
      "dataSample": "LLLLLLLLL",
      "count": 10
    },
    {
      "dataSampleId": 7,
      "executionEventId": 1,
      "dataSample": "LLLLLLLL",
      "count": 40
    }
  ]
}
```

More info

http://<myMaskingEngine>/masking/api-client/#!/non-conformant-data-sample/getAllNon-conformantDataSamples

API Calls Involving File Upload and Download

File Download

API calls involving file download through API client are noteworthy because if the request fails, the API client will continue to show the "loading" icon indefinitely.

To avoid this, make all file download calls through CURL instead. An example of a file download call using CURL is below.

```
curl -X GET --header 'Accept: application/octet-stream' --header
'Authorization: ec443730-124e-4958-a872-324a975bb500'
-o "/home/user/downloads"
'http://<myMaskingEngine>/masking/api/file-downloads/EXPORT-ZXhwb3J0X2RvY3VtZW50X2dGZU9JMVYxLmpzb24%3D'
```

The `-o` flag from above specifies the location to save the file to.

File Upload

API calls involving file upload are noteworthy because the generated curl from the Masking API client will be **missing the parameter referencing the file**; as such, those commands from the Masking API client **will not work**.

Instead, below are examples of working requests and responses for API calls involving file upload.

For commands specific to your masking engine, work with your interactive client at http://**<myMaskingEngine>**/masking/api-client/

> ⚠️ **Warning**
>
> HTTPS (SSL/TLS) is recommended, but for explanatory purposes these examples use insecure HTTP.

> ✏️ **Note**
>
> In all code examples, replace **\<myMaskingEngine>** with the hostname or IP address of your virtual machine.

Creating a File Format

REQUEST

```
curl -X POST --header 'Content-Type: multipart/form-data' --header
'Accept: application/json' --header 'Authorization:
d1313dd8-2ed9-4699-8e88-2b6a089ae2a6' -F
fileFormat=@/path/to/file_format/delimited_format.txt -F
fileFormatType=DELIMITED
'http://<myMaskingEngine>/masking/api/file-formats'
```

RESPONSE

```
{ "fileFormatId": 123, "fileFormatName": "delimited_format.txt",
"fileFormatType": "DELIMITED"
}
```

More info

http://<myMaskingEngine>/masking/api-client/#!/fileFormat/createFileFormat

Creating an SSH Key

REQUEST

```
curl -X POST --header 'Content-Type: multipart/form-data' --header
'Accept: application/json' --header 'Authorization:
d1313dd8-2ed9-4699-8e88-2b6a089ae2a6' -F
sshKey=@/path/to/ssh_key/this_file_name_is_your_ssh_key_name.txt
'http://<myMaskingEngine>/masking/api/ssh-keys'
```

RESPONSE

```
{ "sshKeyName": "this_file_name_is_your_ssh_key_name.txt"
}
```

More info

http://<myMaskingEngine>/masking/api-client/#!/sshKey/createSshKey

Backwards Compatibility API Usage

> ✏ **Note**
>
> In all examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

In all examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

API Versioning Context

The Masking API is versioned in accordance with the Semantic Versioning format: http://semver.org/. When the Masking API is updated, a new API version will be released. Scripts must reference an explicit API version or else there are no guarantees that the scripts will work on future releases of the Masking API.

Pinning Down a Version Number To Guarantee Backwards-Compatibility

'http://<myMaskingEngine>/masking/api/v5.0.0/environments'

This is the format for specifying a version in the URL of an API request targeting the **environments** endpoints.

Specifying the version for endpoint guarantees that the requester receives a response containing all of the fields that were present in that version of the API. This is intended to allow scripts that specify a masking API version in the URL to continue working upon future upgrades of the Masking product--even if a newer version of the API is available in the future Masking product.

For example, consider the scenario where a script is being developed today with a pinned down version **v5.0.0** in the URL of the API requests. Upon upgrade to a future release of the Masking product that has the API **v5.1.0** available, the same, untouched script that was developed with the pinned down version **v5.0.0** in the URL of the API requests are expected to continue working. That said, in order to leverage any new features of the API **v5.1.0**, the original script will need to be updated to specify the new API version in the URL, and the requests may need to be updated to conform to the new API specification.

While specifying a version for endpoint guarantees that all fields present in that version will be contained in the API response, it does **not** mean that new fields that have since been added to that endpoint in subsequent versions will be excluded. We, therefore, recommend that API users write their scripts to parse the JSON response objects by key name, rather than by key index, to prevent these additional fields from breaking any scripts.

Omitted Version Numbers

'http://<myMaskingEngine>/masking/api/environments'

This is the format for not specifying a version in the URL of an API request targeting the **environments** endpoints. When the API version number is omitted, the latest API version is taken as a default. In the first 5.2 release, an API request with an omitted version number will be interpreted as a request against the **v5.0.0** version of the API. In a future release that hypothetically has the API **v5.3.0** available, an API request with an omitted version number will be interpreted as a request against the **v5.3.0** version of the API.

Scripts that omit the version of the Masking API in the URL are not guaranteed to work upon future upgrades of the Masking product because the API specification may change between versions and requests that conform to the old API specification may not work on the new API specification.

DefaultApiVersion

If the version is omitted from the base path of the request's URL, but wishes to be treated using a specific masking API version that is not the latest version, set the DefaultApiVersion application setting. If the DefaultApiVersion is not set and the version is omitted from the URL, the latest version of the API on that engine will be used.

> ✏️ **Note**
>
> The DefaultApiVersion application setting will not be applied to any requests made from within the masking engine. This mean that the UI, api-client, and phone home will always use the latest API version supported on the engine.

API Response Escaping

In Masking API responses, a backslash character (`\`) is escaped with an additional backslash character (`\\`). Special attention should be paid to this behavior in scenarios where an API response is passed to another system as an input, for example, an automation system.

In such cases, a response might need special handling to convert the double backslash sequence (`\\`) back to a single backslash (`\`).

For example, consider the `POST /ssh-key` API for creating/installing an SSH Key. The result when the `POST /ssh-key` API is called with a file name that contains `\`, such as `\key.txt`, is shown below.

Response Body:

```
{
  "errorMessage": "SSH Key file name should not contain [\\, ;, %, ?, :]"
}
```

API Calls for Managing Masking Job Driver Support Tasks

Enabling driver support tasks is possible for built-in Oracle and MSSQL connectors as well as extended connectors that have a JDBC driver that uses a driver support plugin at the following endpoints:

- Masking jobs - `POST /masking-jobs` and `PUT /masking-jobs/{maskingJobId}`

- Reidentification jobs - `POST /reidentification-jobs` and `PUT /reidentification-jobs/{reidentificationJobId}`

- Tokenization jobs - `POST /tokenization-jobs` and `PUT /tokenization-jobs/{tokenizationJobId}`

Disabling driver support tasks is possible for built-in Oracle and MSSQL connectors as well as extended connectors that have a JDBC driver that uses a driver support plugin at the following endpoints:

- `PUT /masking-jobs/{maskingJobId}`

- `PUT /reidentification-jobs/{reidentificationJobId}`

- `PUT /tokenization-jobs/{tokenizationJobId}`

> **ℹ Info**
>
> The order of the tasks returned in `enabledTasks` in the Job APIs' responses is not indicative of the task execution order. The task order is determined by the order the tasks are added to `getTasks` in the Driver Support Plugin implementation.

The following instructions to enable driver support tasks on an Oracle masking job can be used to enable driver support tasks for applicable reidentification and tokenization jobs as well.

View the Tasks Implemented By Driver Support Plugin

1. Select `GET /plugin` (or `GET /plugin/{pluginId}` if the plugin ID of the driver support is known).

2. Change `` `pluginType `` query parameter to `DRIVER_SUPPORT` (default is `EXTENDED_ALGORITHM`).

3. The response should include the full list of driver support plugins on the masking engine. If the engine only has the builtin Oracle driver support plugin installed, the response will look as follows:

```
{
    "_pageInfo": {
        "numberOnPage": 1,
        "total": 1
    },
    "responseList": [
        {
            "pluginId": 8,
            "pluginName": "dlpx-oracle-driver-support",
            "pluginAuthor": "Delphix Engineering",
            "pluginType": "DRIVER_SUPPORT",
            "originalFileName": "delphix-oracle-driver-support-plugin-1.0.0.jar",
            "originalFileChecksum": "17b06f2fd888888e26a634d501b4ac9be5a91a7f50000a995934145c7afe7e12",
            "installDate": "2021-10-24T18:08:50.868+00:00",
            "builtIn": true,
            "pluginVersion": "1.0.0",
            "description": "This plugin provides built-in driver support functionality for the Oracle JDBC
driver that ships with the Delphix Masking Engine.",
            "pluginObjects": [
                {
                    "objectIdentifier": "1",
                    "objectName": "Disable Constraints",
                    "objectType": "DRIVER_SUPPORT_TASK"
                },
                {
                    "objectIdentifier": "2",
                    "objectName": "Drop Indexes",
                    "objectType": "DRIVER_SUPPORT_TASK"
                },
                {
                    "objectIdentifier": "3",
                    "objectName": "Disable Triggers",
                    "objectType": "DRIVER_SUPPORT_TASK"
                }
            ]
        }
    ]
}
```

Create Masking Job That Enables Tasks

> **ℹ Info**
>
> This assumes a ruleset using the desired connector already exists. The following example demonstrates the creation of an in-place masking job on a built-in Oracle connector. This also assumes you know the ID of the task that you want to enable and have execute as part of a given masking job. To enable tasks to execute as part of a masking job on an *extended* connector, you need to ensure the ruleset points to an extended connector that is using a JDBC driver with a driver support and include the property `enabledTasks` in your request.

1. Select `POST /masking-jobs` to create a masking job using the ruleset you created earlier that targets the desired connector.

2. Format the request body as follows to enable Disable Constraints, Drop Indexes and Disable Triggers per the `objectIdentifier` values returned from the GET Plugin API endpoint:

```
{
    "jobName": "Oracle IP job",
    "rulesetId": 1,
    "jobDescription": "Job description",
    "enabledTasks": [
        {
            "taskId": 1
        },
        {
            "taskId": 2
        },
        {
            "taskId": 3
        }
    ]
}
```

The response will look similar to the following with a return status of 200:

```
{
    "maskingJobId": 1,
    "jobName": "Oracle IP job",
    "rulesetId": 1,
    "rulesetType": "table",
    "createdBy": "admin",
    "createdTime": "2021-04-27T21:29:46.043+00:00",
    "feedbackSize": 50000,
    "jobDescription": "Job description",
    "maxMemory": 1024,
    "minMemory": 1024,
    "multiTenant": false,
    "numInputStreams": 1,
    "onTheFlyMasking": false,
    "databaseMaskingOptions": {
        "batchUpdate": true,
        "commitSize": 10000,
        "disableConstraints": false,
        "dropIndexes": false,
        "disableTriggers": false,
        "numOutputThreadsPerStream": 1,
        "truncateTables": false
    },
    "failImmediately": false,
    "enabledTasks": [
        {
            "taskId": 1
        },
        {
            "taskId": 2
        },
        {
            "taskId": 3
        }
    ],
    "streamRowLimit": 20000
}
```

Disable Tasks

To disable the Disable Triggers task on an Oracle masking job, the request body to `PUT /masking-jobs/1` should
exclude the taskId of the task to disable. Using the above request body as an example, Disable Triggers has a task ID
of 3 so the request body to `PUT /masking-job/1` should exclude the object in enabledTasks with `"taskId": 3`. The
request body should thus be:

```
{
    "jobName": "Oracle IP job",
    "rulesetId": 1,
    "jobDescription": "Job description",
    "onTheFlyMasking": false,
    "enabledTasks": [
        {
            "taskId": 1
        },
        {
            "taskId": 2
        }
    ]
}
```

The Oracle masking job will now only have Disable Constraints and Drop Indexes enabled (in this example, their respective task IDs are 1 and 2). The response will look similar to the following with a return status of 200:

```
{
    "maskingJobId": 1,
    "jobName": "Oracle IP job",
    "rulesetId": 1,
    "rulesetType": "table",
    "createdBy": "admin",
    "createdTime": "2021-04-27T21:29:46.043+00:00",
    "feedbackSize": 50000,
    "jobDescription": "Job description",
    "maxMemory": 1024,
    "minMemory": 1024,
    "multiTenant": false,
    "numInputStreams": 1,
    "onTheFlyMasking": false,
    "databaseMaskingOptions": {
        "batchUpdate": true,
        "commitSize": 10000,
        "disableConstraints": false,
        "dropIndexes": false,
        "disableTriggers": false,
        "numOutputThreadsPerStream": 1,
        "truncateTables": false
    },
    "failImmediately": false,
    "enabledTasks": [
        {
            "taskId": 1
        },
        {
            "taskId": 2
        }
    ],
    "streamRowLimit": 20000
}
```

API Examples

loginCredentials

```bash
#!/bin/bash

#
# This file contains all the login information for the masking engine.
#

# Login credentials for the Masking Engine.
USERNAME="myUsername"
PASSWORD="myPassword"

# Login into a masking engine
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '\{ \
"username": "myUsername", \ "password": "myPassword" \ }' 'http://<myMaskingEngine>/masking/api/login'
```

helpers

```bash
#!/bin/bash

#
# This file contains helpers for the various Masking API cookbook scripts.
# This script uses jq to process JSON. More information can be found here - https://stedolan.github.io/jq/.
#

# Login and set the correct $AUTH_HEADER.
login() {
    echo "* logging in..."
    LOGIN_RESPONSE=$(curl -s $SSL_CERT -X POST -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/login <<EOF
{
  "username": "$USERNAME",
  "password": "$PASSWORD"
}
EOF)
    check_error "$LOGIN_RESPONSE"
    TOKEN=$(echo $LOGIN_RESPONSE | jq -r '.Authorization')
    AUTH_HEADER="Authorization: $TOKEN"
}

# Get all applications and select the first one. Place the applicationName in $APPLICATION_ID.
get_application_id() {
    echo "* getting all applications and selecting first one"
    APPLICATIONS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' $MASKING_ENGINE/applications)
    check_error "$APPLICATIONS_RESPONSE"
    NUM_APPLICATIONS=$(echo $APPLICATIONS_RESPONSE | jq -r '._pageInfo.total')
    check_empty $NUM_APPLICATIONS "found no applications to use"
    APPLICATION_ID=$(echo $APPLICATIONS_RESPONSE | jq -r '.responseList[0].applicationName')
    echo "using application '$APPLICATION_ID'"
}
# Get all environments and select the first one. Place the environmentId in $ENVIRONMENT_ID.
get_environment_id() {
    echo "* getting all environments and selecting first one"
    ENVIRONMENTS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' $MASKING_ENGINE/environments)
    check_error "$ENVIRONMENTS_RESPONSE"
    NUM_ENVIRONMENTS=$(echo $ENVIRONMENTS_RESPONSE | jq -r '._pageInfo.total')
    check_empty $NUM_ENVIRONMENTS "found no environments to use"
    ENVIRONMENT_ID=$(echo $ENVIRONMENTS_RESPONSE | jq -r '.responseList[0].environmentId')
    echo "using environment '$ENVIRONMENT_ID'"
}
# Get all database connectors and select the first one. Place the databaseConnectorId in $CONNECTOR_ID.
get_connector_id() {
    echo "* getting all database connectors and selecting first one"
    CONNECTORS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' $MASKING_ENGINE/database-connectors)
    check_error "$CONNECTORS_RESPONSE"
    NUM_CONNECTORS=$(echo $CONNECTORS_RESPONSE | jq -r '._pageInfo.total')
    check_empty $NUM_CONNECTORS "found no db connectors to use"
    CONNECTOR_ID=$(echo $CONNECTORS_RESPONSE | jq -r '.responseList[0].databaseConnectorId')
    echo "using database connector '$CONNECTOR_ID'"
}

# Get all database rulesets and select the first one. Place the databaseRulesetId in $RULESET_ID.
get_ruleset_id() {
    echo "* getting all database rulesets and selecting first one"
    RULESETS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json'
```

```
$MASKING_ENGINE/database-rulesets)
    check_error "$RULESETS_RESPONSE"
    NUM_RULESETS=$(echo $RULESETS_RESPONSE | jq -r '._pageInfo.total')
    check_empty $NUM_RULESETS "found no db rulesets to use"
    RULESET_ID=$(echo $RULESETS_RESPONSE | jq -r '.responseList[0].databaseRulesetId')
    echo "using database ruleset '$RULESET_ID'"
}
# Get all database tables for a database connector specified by $CONNECTOR_ID. Select the first one and
place in $TABLE_NAME.
get_table() {
    echo "* getting all tables for connector '$CONNECTOR_ID' and selecting first one"
    TABLES_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json'
$MASKING_ENGINE/database-connectors/$CONNECTOR_ID/fetch)
    check_error "$TABLES_RESPONSE"
    NUM_TABLES=$(echo $TABLES_RESPONSE | jq -r '. | length')
    check_empty $NUM_TABLES "found no tables to use"
    TABLE_NAME=$(echo $TABLES_RESPONSE | jq -r '.[0]')
    echo "using table '$TABLE_NAME'"
}


# Get all column metadata for table metadata specified by $TABLE_METADATA_ID. Select the first one and
place in $COLUMN_METADATA_ID.
get_column_metadata_id() {
    echo "* getting all column metadata belonging to table metadata '$TABLE_METADATA_ID' and selecting the
first one"
    COLUMNS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json'
$MASKING_ENGINE/column-metadata?table_metadata_id=$TABLE_METADATA_ID)
    check_error "$COLUMNS_RESPONSE"
    NUM_COLUMNS=$(echo $COLUMNS_RESPONSE | jq -r '. | length')
    check_empty $NUM_COLUMNS "found no columns to use"
    COLUMN_METADATA=$(echo $COLUMNS_RESPONSE | jq -r '.responseList[0]')
    COLUMN_METADATA_ID=$(echo $COLUMN_METADATA | jq -r '.columnMetadataId')
    echo "using column '$COLUMN_METADATA_ID'"
}


# Get all masking jobs and select the first one. Place the jobId in $MASKING_JOB_ID.
get_masking_job_id() {
    echo "* getting all masking jobs and selecting first one"
    MASKINGJOB_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' $MASKING_ENGINE/masking-jobs)
    check_error "$MASKINGJOB_RESPONSE"
    NUM_MASKINGJOB=$(echo $MASKINGJOB_RESPONSE | jq -r '._pageInfo.total')
    check_empty $NUM_MASKINGJOB "found no masking jobs to use"
    MASKING_JOB_ID=$(echo $MASKINGJOB_RESPONSE | jq -r '.responseList[0].maskingJobId')
    echo "using masking job '$MASKINGJOB_ID'"
}


# Check if $1 is equal to 0. If so print out message specified in $2 and exit.
check_empty() {
    if [ $1 -eq 0 ]; then
        echo $2
        exit 1
    fi
}


# Check if $1 is an object and if it has an 'errorMessage' specified. If so, print the object and exit.
check_error() {
    # jq returns a literal null so we have to check against that...
    if [ "$(echo "$1" | jq -r 'if type=="object" then .errorMessage else "null" end')" != 'null' ]; then
        echo $1
        exit 1
```

```
    fi
}
```

apiHostInfo

```bash
#!/bin/bash

#
# This file contains all the host information for the masking engine. Additionally,
# this file allows configuration of SSL if desired.
#


# update host name
HOST="myMaskingEngine.com"
API_PATH="masking/api"

# To connect via SSL, set $SSL to "on" and update the port if necessary (default 8443).
# Additionally, you must update the path to the ssl certificate.
SSL="off"
SSL_PORT="8443"
# update cert name
SSL_CERT_PATH="self-signed.cer"

if [ "$SSL" = "on" ]
then
    MASKING_ENGINE="https://$HOST:$SSL_PORT/$API_PATH"
    SSL_CERT="--cacert $SSL_CERT_PATH"
else
    MASKING_ENGINE="http://$HOST/$API_PATH"
    SSL_CERT=""
fi
```

Configure enclosure escape character

```bash
#!/bin/bash

#
# This script is an "out of the box" script that goes through
# Login and configure the enclosure escape character using PUT /file-metadata API
# with the authentication token from Login.
# User need to set DOUBLE_ENCLOSURE, CUSTOM_ENCLOSURE_ESCAPE_CHARACTER and RULESET_ID accordingly
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

# Set DOUBLE_ENCLOSURE=true if you want to set enclosure escape character same as enclosure character,
# and if DOUBLE_ENCLOSURE=true then CUSTOM_ENCLOSURE_ESCAPE_CHARACTER value will be ignored.
DOUBLE_ENCLOSURE=true
# Replace * with your custom escape character if you want to set custom enclosure escape character
# and also DOUBLE_ENCLOSURE=false need to set
CUSTOM_ENCLOSURE_ESCAPE_CHARACTER="\"*\""
# Comment this RULESET_ID if you want to update for all delimited file ruleset for which enclosure is
defined.
#RULESET_ID=1

echo "Calling GET /file-metadata API"
if [[ -z "$RULESET_ID" ]] || [ "$RULESET_ID" = "null" ] || [ "$RULESET_ID" = "" ]; then
    FILE_METADATA_RESPONSE=$(curl $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Accept: application/json'
''"$MASKING_ENGINE/file-metadata"'')
else
    FILE_METADATA_RESPONSE=$(curl $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Accept: application/json'
''"$MASKING_ENGINE/file-metadata?ruleset_id=$RULESET_ID"'')
fi

i=0
while true; do
    ENCLOSURE=$(jq '.responseList['$i'] .enclosure' <<<"$FILE_METADATA_RESPONSE")

    if [ "$DOUBLE_ENCLOSURE" = true ]; then
        CUSTOM_ENCLOSURE_ESCAPE_CHARACTER=$ENCLOSURE
    fi

    UPDATED_FILE_METADATA_RESPONSE=$(jq '.responseList['$i']
.enclosureEscapeCharacter='$CUSTOM_ENCLOSURE_ESCAPE_CHARACTER'' <<<"$FILE_METADATA_RESPONSE")
    FILE_METADATA_RESPONSE=$UPDATED_FILE_METADATA_RESPONSE
    FILE_METADATA_OBJECT=$(jq '.responseList['$i']' <<<"$FILE_METADATA_RESPONSE")
    FILE_METADATA_ID=$(jq '.responseList['$i'] .fileMetadataId' <<<"$FILE_METADATA_RESPONSE")

    if [[ -z "$FILE_METADATA_ID" ]] || [ "$FILE_METADATA_ID" = "null" ]; then
        break
    else
        if [[ ! -z "$ENCLOSURE" ]] && [ ! "$ENCLOSURE" = "null" ] && [ ! "$ENCLOSURE" = "" ]; then
            echo "Calling $MASKING_ENGINE/file-metadata/$FILE_METADATA_ID API to update enclosure escape
character=$CUSTOM_ENCLOSURE_ESCAPE_CHARACTER"
            UPDATE_RESPONSE=$(curl $SSL_CERT -X PUT -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json'
-H 'Accept: application/json' -d ''"$FILE_METADATA_OBJECT"'' ''"$MASKING_ENGINE/file-
metadata/$FILE_METADATA_ID"'')
            check_error "$UPDATE_RESPONSE"
        fi
    fi
```

```
    ((i++))
done

echo
```

createApplication

```bash
#!/bin/bash

#
# This script will login and create an application. It depends on helpers in the helpers script as well as
host and login
# information found in apiHostInfo and loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

echo "* creating application 'App123'..."
curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/applications <<EOF
{
    "applicationName": "App123"
}
EOF

echo
```

createEnvironment

```bash
#!/bin/bash

#
# This script will login and create an environment with an application. It depends on helpers in the
helpers
# script as well as host and login information found in apiHostInfo and loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

#
# When deciding which application to place the environment in we simply choose the first application found.
You are
# encouraged to modify this to suit your needs. Please see get_application_id in helpers for more
information.
#
get_application_id

echo "* creating environment 'newEnv' in application '$APPLICATION_ID'..."
curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/environments <<EOF
{
    "environmentName": "newEnv",
    "application": "$APPLICATION_ID",
    "purpose": "MASK"
}
EOF

echo
```

createInventory

```bash
#!/bin/bash

#
# This script will login, create table metadata for a given table name and ruleset, and then update an
# inventory (i.e. assign an algorithm and domain to a specific column of the table). It depends on helpers
# in the helpers script as well as host and login information found in apiHostInfo and loginCredentials,
respectively.
# This script uses jq to process JSON. More information can be found here - https://stedolan.github.io/jq/.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

#
# When deciding which connector, ruleset, and table to use we simply use the first ones found of each. You
are
# encouraged to modify this to suit your needs. Please see the respective functions in helpers for more
information.
#
get_connector_id
get_ruleset_id
get_table

echo "* creating table metadata for ruleset id '$RULESET_ID' with table '$TABLE_NAME'..."
TABLE_METADATA_RESPONSE=$(curl $SSL_CERT -s -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' -H 'Accept: application/json' --data @- $MASKING_ENGINE/table-metadata <<EOF
{
    "tableName": "$TABLE_NAME",
    "rulesetId": $RULESET_ID
}
EOF)
check_error "$TABLE_METADATA_RESPONSE"
TABLE_METADATA_ID=$(echo $TABLE_METADATA_RESPONSE | jq -r '.tableMetadataId')
echo "using table metadata '$TABLE_METADATA_ID'"

get_column_metadata_id

curl $SSL_CERT -X PUT -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/column-metadata/$COLUMN_METADATA_ID  <<EOF
{
    "algorithmName": "AddrLine2Lookup",
    "domainName": "ADDRESS_LINE2"
}
EOF

echo
```

create DatabaseConnector

```bash
#!/bin/bash

#
# This script will login and create a database connector in an environment. It depends on helpers in the
helpers
# script as well as host and login information found in apiHostInfo and loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

#
# When deciding which environment to place the connector in we simply choose the first environment found.
You are
# encouraged to modify this to suit your needs. Please see get_environment_id in helpers for more
information.
#
get_environment_id

echo "* creating database connector 'connector' in environment '$ENVIRONMENT_ID'..."
curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/database-connectors <<EOF
{
    "connectorName": "connector",
    "databaseType": "ORACLE",
    "environmentId": $ENVIRONMENT_ID,
    "host": "myHost",
    "password": "myPassword",
    "port": 1234,
    "schemaName": "MYSCHEMA",
    "sid": "mySID",
    "username": "MYUSERNAME"
}
EOF

echo
```

create DatabaseRuleset

```bash
#!/bin/bash

#
# This script will login and create a database ruleset for a database connector. It depends on helpers in
the helpers
# script as well as host and login information found in apiHostInfo and loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

#
# When deciding which database connector we will use, we simply choose the first database connector found.
You are
# encouraged to modify this to suit your needs. Please see get_connector_id in helpers for more
information.
#
get_connector_id

echo "* creating database ruleset 'myRuleset' in db connector '$CONNECTOR_ID'..."
curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/database-rulesets <<EOF
{
    "rulesetName": "myRuleset",
    "databaseConnectorId": $CONNECTOR_ID
}
EOF

echo
```

getAuditLogs

```bash
#!/bin/bash

#
# This script is an "out of the box" script that goes through
# Login and GET /audit-logs with the authentication
# token from Login
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

echo "* GET /audit-logs from $EXPORT_ENGINE"
EXPORT_RESPONSE=$(curl $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Accept: application/json'
$MASKING_ENGINE/audit-logs)

# Calculate the number of audit log entries and the proximity to the entry limit.
AUDIT_ENTRY_COUNT=$(jq '._pageInfo.total' <<<"$EXPORT_RESPONSE")
MAX_ENTRIES=1000000
DIFFERENCE=$((MAX_ENTRIES-AUDIT_ENTRY_COUNT))

# Retrieve the date of the oldest audit entry retained.
OLDEST_DATE=$(jq '.responseList[1].activityTime' <<<"$EXPORT_RESPONSE")

echo "There are $AUDIT_ENTRY_COUNT entries in the audit log. After $DIFFERENCE more audits you will hit the
$MAX_ENTRIES limit and will begin to overwrite entries starting from the oldest, which was created on:
$OLDEST_DATE"
```

getSyncableObjects

```bash
#!/bin/bash

#
# This script is an "out of the box" script that goes through
# Login and GET /syncable-objects with the authentication
# token from Login
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

echo "* GET /syncable-objects from $EXPORT_ENGINE"
EXPORT_RESPONSE=$(curl $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Accept: application/json'
$MASKING_ENGINE/syncable-objects)
echo $EXPORT_RESPONSE
```

getSyncableObjectsExport

```bash
#!/bin/bash

#
# This script will log in and get all syncable objects on
# the Masking Engine and then, given a grouping command, save the
# exported document in a file and export all syncable objects
# in the indicated group
#
# Grouping command:
# algoType: -t <LOOKUP | BINARYLOOKUP | SEGMENT | TOKENIZATION | MAPPLET | KEY>
# algoCd: -n <RegexForAlgoName>
#
# Currently the response from GET /syncable-objects is saved
# to getobj_response.json, and the grouped input for /export
# in grouped_export_list.json, and the final export response
# into export_response.json. But of course, this can script
# can be modified to save to other specified places.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

echo "* GET /syncable-objects"
GETOBJ_RESPONSE=$(curl $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json'
$MASKING_ENGINE/syncable-objects)
echo $GETOBJ_RESPONSE > "./getobj_response.json"

# Create a temporary export list file
GROUPED_EXPORT_LIST="./grouped_export_list.json"
echo "[]" > $GROUPED_EXPORT_LIST

if [[ $1 == "-t" ]]; then
    ALGO_TYPE=$2
    echo "* Filter for all syncable objects of algorithm type $ALGO_TYPE"

    jq -c '.responseList[]' getobj_response.json | while read i; do
        if [[ $(echo $i | jq '.objectType') == \"$ALGO_TYPE\" ]]; then
            # The key to getting the correct json format here was to use
            # the --argjson instead of --arg. --arg will stringify everything
            # and escape all special characters like {, ", etc.
            echo $(cat $GROUPED_EXPORT_LIST | jq --argjson obj "$i" '. |= . + [$obj]') > $GROUPED_EXPORT_LIST
        fi
    done
elif [[ $1 == "-n" ]]; then
    ALGO_NAME_REGEX=$2
    echo "* Filter for all syncable objects where algorithmCd matches the regex $ALGO_NAME_REGEX"

    jq -c '.responseList[]' getobj_response.json | while read i; do
        if [[ "$(echo $i | jq '.objectIdentifier.algorithmName')" =~ \"$ALGO_NAME_REGEX\" ]]; then
            echo $(cat $GROUPED_EXPORT_LIST | jq --argjson obj "$i" '. |= . + [$obj]') > $GROUPED_EXPORT_LIST
        fi
    done
fi

echo "* Export syncable objects from $GROUPED_EXPORT_LIST"
EXPORT_RESPONSE=$(curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H
'Accept: application/json' -d "$(<$GROUPED_EXPORT_LIST)" $MASKING_ENGINE/export)
```

```
# Save the grouped export response into a file
echo $EXPORT_RESPONSE > export_response.json
echo '* Completed exporting. Check "export_response.json" for the export document. This export document
json object will be what you literally put in as the input for import'
```

Add a new Type Expression

```bash
#!/bin/bash

#
# This script will login and create a profile type expression. It depends on helpers in the helpers script
as well as host and login
# information found in apiHostInfo and loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/profile-type-expressions <<EOF
{
  "domainName": "FIRST_NAME",
  "expressionName": "FirstNameType",
  "dataType": "String",
  "minDataLength": 5
}
EOF

echo
```

To be effective, a Profile Type Expression has to be part of a profile set. A type expression can be added to a profile set with the profile-sets endpoint. For example, if some Profile Type Expressions were created and have ids 57 and 48, we can use the PUT method on the profile-set endpoint to update an existing profile set so that it includes the new profile type expression. This is shown below, where the profile set has id 42.

```bash
#!/bin/bash

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

curl $SSL_CERT -X PUT -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/profile-sets/42 <<EOF
{
  "profileSetName": "FINDS_ALL_SENSITIVE_DATA",
  "profileExpressionIds": [
    4,
    8,
    12,
    13,
    27
  ],
  "profileTypeExpressionIds": [
    57,
    58
  ]
}
EOF
```

Delete a Type Expression

Deleting a type expression is done using the DELETE method on the profile-type-expression endpoint. The
expression must be removed from any profile sets it's a part of before it can be deleted.

```bash
#!/bin/bash

#
# This script will login and delete a profile type expression. It depends on helpers in the helpers script
as well as host and login
# information found in apiHostInfo and loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

echo "* creating application 'App123'..."
curl $SSL_CERT -X DELETE -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/profile-type-expressions/57

echo
```

runMaskingJob

This script will login and run a masking job. It depends on helpers in the helpers script as well as host and login information found in apiHostInfo and loginCredentials, respectively.

```bash
#!/bin/bash
source apiHostInfo
eval $(cat loginCredentials)
source helpers

login
```

When deciding which masking job to run, we simply choose the first masking job found. You are encouraged to modify this to suit your needs. Please see get_masking_job_id in helpers for more information.

```
get_masking_job_id

echo "* running masking job '$MASKING_JOB_ID'..."
curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type: application/json' -H 'Accept:
application/json' --data @- $MASKING_ENGINE/executions <<EOF
{
    "jobId": "$MASKING_JOB_ID"
}
EOF
echo
```

If a masking job is called by a PowerShell hook script, the following command MUST be added to the script using the **Powershell -File** prefix, **file path;**, and the **exit $LASTEXITCODE** suffix.

```
PowerShell -File C:\Users\HomeFolder\AddUser.ps1; exit $LASTEXITCODE
```

If this is not added then Delphix will not know if the script ran or completed. For more information, please visit this SQL Server PowerShell Script Error Handling documentation.

Authoring Extensible Plugins

Introduction

The SDK was formerly referred to as the *Masking Algorithm SDK*, but it is now referred to as the *Masking Extensible SDK*, as of SDK version 1.5.0, as it now allows for the development of different types of extensible plugins. As of Delphix release 6.0.3.0, the Delphix Masking Engine supports the installation of plugins, written in Java, that provide new masking algorithms; and as of 6.0.9.0, driver support plugins. The former feature is referred to as Extensible Algorithms and the latter is referred to as Extensible Driver Supports. This section of the documentation details all aspects of masking algorithm and driver support plugin usage and development. The *Guided Tour* portion of the workflows section for Extensible Algorithms and Extensible Driver Supports walk the user through the basic process of building a simple plugin and installing it onto the Delphix Masking Engine. Other sections explore in-depth topics such as making algorithms configurable, consuming input files, etc.

This documentation assumes the reader has some familiarity with Java development as well as operation of the Delphix Masking Engine via both the UI and Web API Client. The reader should also understand the security requirements associated with any new algorithms being developed.

Before Getting Started

This documentation assumes you have a functional Java 8 development environment. Instructions for setting up a basic development environment are here.

You should also download the Extensible SDK binary package from the Delphix download site and unpack it into a new directory on your development system. This directory - the root of the unpacked archive - will be referred to as *sdk_root*.

It's helpful to add the binaries directory to your PATH. On a UNIX like system, this command will add the SDK utilities to PATH:

```
$ PATH=$PATH:$(pwd)/sdkTools/bin
```

It is presumed that the SDK bin directory is in the user's PATH throughout this documentation.

SDK Features

The Extensible SDK provides a number of useful functions that aid development of new algorithms and driver supports for the Delphix Masking Engine. It is available on the Delphix software download site.

- Creation of empty "skeleton" projects, with build files - the maskScript *init* sub-command

- Creation of empty class files for algorithms and driver supports - the maskScript *generate* sub-command

- Testing of masking algorithms and driver supports without a masking engine

  - The maskApp CLI (*only algorithms*)

  - The maskScript *mask* sub-command (*both algorithms and driver supports*)

- Uploading of plugins to the masking engine - the maskScript *install* sub-command

- Sample algorithms and driver supports that illustrate the usage of key features of the Masking Plugin API

Versions compatibility

The SDK shares some key elements with the Masking Engine, so in order for the SDK to provide behaviors as close as possible to the Masking Engine, use the SDK version which corresponds to the Masking Engine where you are planning to use the created algorithm(s). The SDK and the Masking Engine use a common Masking API which provides the mechanisms to run the extensible algorithms. Masking algorithms built on the SDK using the latest Masking API will not necessarily run on an older Masking Engine version.

Delphix Release	Masking API*	Extensible SDK*
6.0.3	1.0.0	-
6.0.4	1.1.0	1.0.0
6.0.5	1.1.0	1.1.0
6.0.6	1.2.0	1.2.0
6.0.7	1.3.0	1.3.0
6.0.8	1.4.0	1.4.0
6.0.9	1.5.0	1.5.0

* Note that prior to Delphix Release 6.0.9 and SDK release 1.5.0, Masking API was referred to as Algorithm API and Extensible SDK as Algorithm SDK.

Getting More Information

Several other sources of information are available to aid in plugin development:

- The README.md file under docs in the Extensible SDK download archive

- The Masking Plugin API Javadoc

- Invoke **maskScript** (located under *sdkTools/bin* in the SDK download) with the -h option for usage help

- Type help at the **maskApp** (also under *sdkTools/bin* in the SDK download) command prompt

General Plugin Structure

Introduction

This section describes the structure of the plugin Java archive (JAR) files used to extend the Delphix Masking Engine with additional algorithms. This includes the **MaskingAlgorithm** interface that classes providing new algorithm code must implement, and various other metadata present in the plugin JAR required for the plugin to be usable. It also discusses some aspects of build dependencies and common pitfalls involved when adding new 3rd party dependencies.

Plugins for the Masking Engine should be self-contained. This means they should included all Java classes necessary to run, with a few critical exclusions. The Java classes that comprise the Masking Plugin API itself are the exception; these must be excluded from the plugin JAR to ensure that the plugin properly uses the API classes present on the Masking Engine (or SDK during the test process). This is described in more detail in the dependency management section.

Dependency Management

A vast assortment of third-party Java libraries are available, expanding the set of ready-to-use functionality well beyond what is already a rich standard library. Plugins for the Delphix Masking engine are able to make use of external libraries, but a number of guidelines should be followed to ensure proper function and compatibility. Note that the plugin classloader uses a plugin-first loading strategy for dependencies.

How to Properly Use and Embed External Libraries

When using an external library in a plugin for the Delphix Masking engine, consider these guidelines:

- The plugin JAR should contain all external libraries. This is commonly referred to as a "fat JAR". This prevents the plugin code from inadvertently linking with copies of the same library that might happen to be part of the Masking Engine's codebase, leading to potential version conflicts and unpredictable behavior across upgrades.

- **However**, a small set of packages defining the interface between plugins and the Delphix Masking Engine **must not** be embedded in the JAR. It is critical that, for these packages, the plugin code link against the same classes already loaded by the engine. These packages are:

  - com.delphix.masking:masking-algorithm-api

  - com.fasterxml.jackson.core:jackson-annotations

  - com.google.code.findbugs:jsr305

  - junit:junit

If the externally created plugin uses any of the mentioned libraries, the exact same libraries versions should be used by the plugin author, as the ones used by the SDK. The way to find those versions is:

```
- in the installed SDK find the following gradle file under `samples` directory:
    * gradle.properties
```

It contains the versions of the SDK provided external libraries, for example:

```
        googleGuavaVer=28.0-jre
        maskingAlgoVer=1.3.0
        jacksonVer=2.9.5
        junitVer=4.12
```

Looking to those versions author should decide what version of corresponding library to use (if it is required by their design).

- Plugins consuming third-party libraries should be thoroughly tested, as it is not uncommon that library code will attempt to use permissions not granted by the plugin sandbox. If this is the case, there is currently no way to modify the constraints under which the plugin code is executed.

- The plugin author, not *Delphix*, is responsible for ensuring that any license files or other forms of attribution required by any embedded software are handled properly.

- The entity deploying the plugin, not *Delphix*, is responsible for ensuring the organization operating the Masking Engine has obtained the necessary licenses or rights to use any embedded software.

Example Build File

The following fragments, derived from the sample algorithm *build.gradle* file, illustrate how to correctly build a plugin using the gradle build system:

```
jar {
    from {
        configurations.runtimeClasspath.collect { it.isDirectory() ? it : zipTree(it) }
    }
    includeEmptyDirs = false

    manifest {
        attributes(
                (PluginMetadata.PLUGIN_NAME_KEY)       : "SampleAlgorithms",
                (PluginMetadata.AUTHOR_NAME_KEY)       : "Sample Author,
                (PluginMetadata.PLUGIN_VERSION_KEY)    : "1.0.0 ${getGitHash}",
                (PluginMetadata.ALGORITHM_API_VERSION_KEY): maskingAlgoVer,
                'Build-Timestamp': new java.text.SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZ").format(new
Date()),
                'Created-By'     : "Gradle ${gradle.gradleVersion}",
                'Build-Jdk'      : "${System.properties['java.version']}
(${System.properties['java.vendor']} ${System.properties['java.vm.version']})",
                'Build-OS'       : "${System.properties['os.name']} ${System.properties['os.arch']}
${System.properties['os.version']}",
        )
    }
}

dependencies {
    compileOnly ('com.google.code.findbugs:jsr305:3.0.2')
    compileOnly ('com.delphix.masking:masking-algorithm-api:' + maskingAlgoVer)
    compileOnly ('com.fasterxml.jackson.core:jackson-annotations:' + jacksonVer)

    compile 'com.google.guava:guava:' + googleGuavaVer

    testImplementation 'com.google.code.findbugs:jsr305:3.0.2'
    testImplementation 'com.delphix.masking:masking-algorithm-api:' + maskingAlgoVer
    testImplementation 'com.fasterxml.jackson.core:jackson-annotations:' + jacksonVer
    testImplementation 'junit:junit:' + junitVer
    testImplementation "com.google.truth:truth:" + googleTruthVer
}
```

How this works:

- The "from { ... }" property of the **jar** section instructs gradle to included all classed need at runtime in the plugin JAR file.

- In the **dependencies** section, packages comprising the interface between the plugin and Masking Engine are listed as *compileOnly*. This excludes them from the runtime environment and causes them to be omitted from the plugin JAR file.

- The third-party code dependency on the popular Google Guava library is listed as *compile*, causing it to be included in the plugin JAR file.

> **✎ Note**
>
> Variables defining the package dependency versions are typically read from the *gradle.properties* file.

Plugin Metadata

When a plugin is built for use with the Masking Engine, it is critical that certain metadata be included in the plugin JAR file. This includes certain attributes in the JAR's manifest, as well as the service discovery file used to determine which classes in the JAR are directly usable by the Extensibility Framework.

Manifest Attributes

Java archives carry metadata attributes in the manifest file, located at *META-INF/MANIFEST.MF* in the archive file. Some of these attributes are required or at minimum quite useful in a plugin's manifest.

The following attributes carry special meaning to the extensibility framework when present in a plugin's manifest. Care must be taken to ensure they are set to valid and meaningful values for any plugins intended for production use. Additional attributes may be supported in the future. Any future attributes introduced for anything beyond a purely informational purpose will be of the format "Delphix-*" to avoid conflict with any preexisting usage.

Recognized Manifest Attributes

Attribute	Meaning	Example Value
Delphix-Plugin-Name	The default name of the plugin. This name will be used on the Masking Engine unless overridden at plugin install time. All plugin names beginning with the string "dlpx" are reserved for future use by modules delivered with the Delphix Masking Engine product.	SamplePlugin
Implementation-Vendor	The individual or organization that authored the plugin module.	Sample Inc.
Implementation-Version	The version of the plugin. This is an entirely free-form string, limited to 255 characters.	1.0.0-SNAPSHOT
Delphix-Algorithm-API-Version	The version of the Delphix Masking Plugin API used by the plugin. This value **must** be present and represent a valid API version.	1.0.0

> ✏️ **Note**
>
> The maskScript *init* sub-command adds logic to the gradle build files to ensure that the project build inserts the correct attribute values into the plugin manifest.

Versioning

In order to ensure compatibility between each software component in the extensible algorithms system, careful use of versioning must be enforced. The goals are twofold; first, to ensure that the version of each software module is visible, and second, to ensure that incompatible plugins are detected and prevented from running. The software modules particular to extensible algorithms - the Masking Plugin API and Masking Algorithm SDK, use a version number in the following format: Major.Minor.Micro, with an option -*TEXT* notation.

Version information for the plugin, including the plugin version and the version of the Masking Plugin API it was built against, are embedded in the plugin JAR file as metadata.

Table of Versioned Objects

The following table explains how each software module is versioned, and what enforcement takes place:

Software Module	Example Values	Details
Delphix Masking Engine	6.0.3.0	None, however the Masking Algorithm API version is fixed for each particular Delphix Masking Engine release.
Masking Plugin API	1.0.0	The version of the Masking Plugin API used to build a plugin must be embedded in each Plugin. This is done automatically when plugins are built using the Masking Algorithm SDK. Currently, as only one version of the Masking Plugin API exists, the only enforcement in place ensures that a valid version has been been embedded in the plugin metadata. In the future, should it be necessary to make backward incompatible changes to the Masking Plugin API, a support matrix will be established and enforced.
Masking Algorithm SDK	1.0.0	Each version of the Masking Algorithm SDK will have a maximum version of the Masking Plugin API it can handle, and will refuse to work with future versions. This is not currently enforced.
Plugins	Author defined	No enforcement. The plugin version will be visible using the Masking API GET operation on the *Plugin* endpoint.

Ensuring Plugin Compatibility

As the implementation of a plugin evolves, it's natural for the software to change. Changes like bug fixes, performance improvements, etc. can typically be made transparently, without endangering backward compatibility. Challenges can arise when it is necessary to change the configuration schema for a framework within a plugin or remove certain algorithm frameworks or instances from a plugin. This section will detail some best practices for maximizing backward compatibility in each case.

In this case, backward compatibility means that it is possible to upgrade the plugin to a new version on the masking engine without removing the previous version of the plugin. While it is always possible to replace a plugin by removing the old plugin and installing a new one, this requires manual reconstruction of any inventory that references the algorithms based on the old plugin, which can be very labor-intensive.

Schema Changes

A schema change means altering the set of configuration parameters exposed by an algorithm framework. For example, this might be done to add a case-insensitive flag to an algorithm that processes Strings. In order to make this kind of change while preserving backward compatibility, the following rules must be followed:

- Existing configurable variables cannot be removed or modified. These are the class fields with to which the @JsonProperty annotation has been applied.

- New configurable variables may be added, but they must have a default value, so that applying a JSON document lacking a value for the new field results in a valid instance.

If changes must be made that do not meet these requirements, it may be preferable to expose the new or modified functionality as a new algorithm framework, rather than changing the existing one.

Component Removal

Component removal means removing an algorithm framework or a built-in instance provided by an existing framework. When this happens, updating to the new version of the plugin will be blocked if any of the removed objects are in use by the Delphix Masking Engine. This includes references in Inventory, Domains, and any File Formats. In addition, the presence of any user-created algorithms based on a framework which is removed in the new plugin version will block updating.

Plugin Naming

One last compatibility concern is the potential for a plugin name to clash with the name of plugins delivered by the Delphix Masking Engine product. Such a clash would make it impossible to upgrade the engine to a new version without first removing the conflicting user installed plugin. To avoid this concern, avoid embedding any default plugin name beginning with the string "dlpx".

Setting Up Your Development Environment

This section describes the step-by-step process for setting up the development environment that was used to develop and test many of the procedures in this Guided Tour. A rich set of tools exist to support Java development, so this is by no mean the only development environment possible.

> ✏️ **Note**
>
> As of version 6.0.3.0, the Delphix Masking Engine's JVM version is 8, so it's important to build a plugin compatible with Java 8.

Downloading and Installing Tools

- Download and install the Oracle Java JDK or OpenJDK from the AdoptOpenJDK Project. Make sure you install **Java 8** version.

- Download and install IntelliJ IDEA Community Edition for your OS. These instructions are known to work for version 2019.3.

Creating a new Project

Identify the root directory of your project code. For example, if you used the instructions to create a new algorithm or driver support project, this directory is referred to as *proj_dir*.

Start IntelliJ IDEA. A pop-up should appear.

1. Select **Project Settings > Project > Project SDK > New > JDK**

2. Provide the path to the JDK you installed earlier (e.g. /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home). Select **OK**.

3. For **Project language level**, set to "8 - Lambda, type annotations etc."

4. In the IntelliJ IDE, select **Import Project** from the pop-up.

5. Provide the path to the root of your project, for example, *proj_dir*.

6. Select **Import project from external model** and **Gradle**, and then select **Next**.

7. After the project is imported, a directory tree is shown on the left panel.

At this point, the development environment should be ready to use.

Enabling Remote Debugging

It is often useful to enable remote debugging, which allows the IDEs debugger to attach to a running **maskApp** or **maskScript** process. To enable remote debugging, certain environment variables must be set. In both cases, the value *5005* can be replaced with the value of any open TCP port.

For **maskApp**

```
MASK_APP_OPTS='-Xdebug -Xnoagent -Djava.compiler=NONE -
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005'
```

For **maskScript**

```
MASK_SCRIPT_OPTS='-Xdebug -Xnoagent -Djava.compiler=NONE -
Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005'
```

> **✎ Note**
>
> These settings will cause the maskScript to suspend at startup to allow time to attach the debugger.

Algorithms

Introduction

As of release 6.0.3.0, the Delphix Masking Engine supports the installation of plugins, written in Java, that provide new masking algorithms. This feature is referred to as Extensible Algorithms. This section of the documentation details all aspects of masking algorithm plugin usage and development. The *Guided Tour* portion of the workflows section walks the user through the basic process of building a simple plugin and installing it onto the Delphix Masking Engine. Other sections explore in-depth topics such as making algorithms configurable, consuming input files, etc.

This documentation assumes the reader has some familiarity with Java development as well as operation of the Delphix Masking Engine via both the UI and Web API Client. The reader should also understand the security requirements associated with any new algorithms being developed.

The Extensible Algorithms framework is designed to replace the custom algorithm (aka. mapplets) feature by providing richer functionality, greatly simplifying algorithm development, and ensuring long-term maintainability of plugins. That said, no specific timeline for the end-of-support of custom algorithms has been announced.

SDK Features

The Masking Algorithm SDK provides a number of useful functions that aid development of new algorithms for the Delphix Masking Engine. It is available on the Delphix software download site.

- Creation of empty "skeleton" projects, with build files - the maskScript *init* sub-command
- Creation of empty class files for algorithms - the maskScript *generate* sub-command
- Testing of masking algorithms without a masking engine
  - The maskApp CLI
  - The maskScript *mask* sub-command
- Uploading of plugins to the masking engine - the maskScript *install* sub-command
- Sample algorithms that illustrate the usage of key features of the Masking Plugin API

Getting More Information

Several other sources of information are available to aid in plugin development:

- The README.md file under docs in the Algorithm SDK download archive
- The Masking Plugin API Javadoc
- Invoke **maskScript** (located under *sdkTools/bin* in the SDK download) with the -h option for usage help
- Type help at the **maskApp** (also under *sdkTools/bin* in the SDK download) command prompt

The MaskingAlgorithm Java Interface

Any Java class that should be recognized as a masking algorithm (whether stand-alone or configurable) must implement the **MaskingAlgorithm** interface. This interface is parameterized with the data type the algorithm masks, which defines the input and output data type of the **mask** method. The full details of this interface are described in the Masking Plugin API Javadoc

Core Data Types

The Delphix Masking Engine is designed to support a wide and extensible set of data sources, which naturally encode data in a variety of different formats. In order to simplify algorithm development, while maintaining the ability to mask data from many sources, we've identified a core set of data formats which are likely to require different masking treatment and ensured that the Extensible Algorithm framework converts all data to/from these types as needed. These types define the allowed parameterization of the **MaskingAlgorithm** Java interface.

Each masking algorithm class is defined to mask exactly one of the following data types:

- Binary data - **java.nio.ByteBuffer**

- String data - **java.lang.String**

- Numeric data - **java.math.BigDecimal**

- Date time data - **java.time.LocalDateTime**

- Multi-column data - **com.delphix.masking.api.plugin.utils.GenericDataRow** (See Multi-Column Masking section)

Each algorithm is expected to input, process, and emit objects of one of the above Java types, but is free to use any intermediate types as needed to access library methods. Because it is frequently the case that data of one type is stored in databases or documents in a type other than its most natural native type (ex. dates stored in VARCHAR fields, or numbers stored as text in a CSV file), the masking framework that executes these algorithms is capable of performing a number of automatic type conversions, detailed in the next section. This allows algorithms written to process one data type to handle data of other types, with no additional work required of the algorithm author.

Supported Automatic Type Conversions

Algorithm Native Type	Supported Type	Notes
ByteBuffer	String	Algorithm receives the UTF-8 encoded value of the String and is expected to return a valid UTF-8 ByteBuffer.
LocalDateTime	String	The correct date format must be assigned to the field or column in the masking inventory.

LocalDateTime	Compatible numeric types	A compatible date format, such as *yyyyMMdd*, must be assigned to the column in inventory.
BigDecimal	All numeric types	Upconverted to BigDecimal. Out of range values after masking are truncated to fit the range of the underlying type.
BigDecimal	String	String value is converted to a number.

Special Case Values

In order to allow algorithms to implement special handling for null, empty, and special case values, these values are presented to the masking algorithm unmodified. Algorithms should be prepared to process the full range of input values possible for the input type. In practice, this means that most **mask** method implementations will begin with a null check on the *input* value, prior to attempting to use the input - for example, by calling **input**.*length()* or similar. It is perfectly acceptable and commonplace to return null in the case where the mask input is null.

Method Overview

This section provides a high-level overview of the methods in the **MaskingAlgorithm** interface. For complete details, consult the Masking Plugin API Javadoc included in the Algorithm SDK archive.

- *getName* and *getDescription* - These methods are used to determine the name and description of frameworks and algorithm instances included in the plugin. For user-created instances, these methods are never called.

- *getDefaultInstances* and *getAllowFurtherInstances* - These methods control the set of instances of the algorithm framework that are defined by the plugin, and whether the user should be allowed to create additional instances.

- *validate* - This method is called after configuration is applied to allow the algorithm class to check whether the injected configuration is valid.

- *setup* and *tearDown* - These methods are called before the algorithm object is used for masking, and after, respectively. Typically, any resources, such as input files, are acquired during *setup* and released during *tearDown*.

- *mask* - This is the method that does the actual data masking in the algorithm class. The input and output values are parameterized for type safety as described above

- *maskBatch* - This method is called to perform masking in situations when it is possible for the caller to build a collection of input values to mask in a single method call. A default implementation is provided that simply calls the mask method on each value in the batch.

- *listMaskedFields* - This method needs to be implemented for Multi-Column Algorithms. It returns a map of field names (`String`) to the Core Data Type. This method does not need to be implemented if not implementing a Multi-Column Algorithm.

- *listReadOnlyFields* - Similar to `listMaskedFields` but optional for Multi-Column Algorithms. Fields returned by this method are read-only and cannot be changed.

The Life Cycles of Algorithm Objects

The Extensibility framework uses objects classes implementing **MaskingAlgorithm** interface for several distinct purposes. These object life cycles are as follows:

Plugin Discovery

This occurs when the extensibility framework evaluates the capabilities present in a **MaskingAlgorithm** class.

1. Java object creation - an object of the algorithm class is created

2. *getName* - determines framework name

3. *getDescription* - determines framework description

4. *getDefaultInstances* - determines all plugin-provided algorithm instances. For each instance:

    a. *getName* - determines instance name

    b. *getDescription* - determines instance description

    c. *validate* - ensure object passes validation

    d. Serialize configurable fields - these are saved as a JSON document defining the instance's configuration

    e. Disposal - the Java object is discarded

5. *getAllowFurtherInstances* - determines whether the framework is visible in the *algorithm/framework* API endpoint

6. Disposal - the Java object is discarded

User Algorithm Creation

This life cycle occurs whenever a user attempts to create a new instance of a plugin algorithm framework. The algorithm definition is saved only if each step succeeds.

1. Java object creation - an object of the algorithm class is created

2. Configuration injection - the values in the user-provided JSON document are injected into the object

3. *validate* - the object's *validate* method is called

4. Disposal - the Java object is discarded

> ✏️ **Note**
>
> The *setup* method is not executed when a user-defined instance is created.

Algorithm Use

This is the life cycle of an algorithm object when used to mask data.

1. Java object creation - an object of the algorithm class is created

2. Configuration injection - the saved JSON document defining this instance is injected in the object

3. *setup* - the *setup* method is called once

4. *mask* - the *mask* method is called on each value to be masked

5. *tearDown* - the *tearDown* method is called once

6. Disposal - the Java object is discarded

> 🔥 **Tip**
>
> It should be noted that a distinct Java object is created for each application of a masking algorithm during Job execution. For algorithms that create or load a large amount of state, this can result in significant memory usage storing redundant data for each instance. This can be avoided using a class level static cache to store data; the instance name, which can be retrieved during *setup* from the **ComponentService** interface object, can be used as an access key for data cached in this way.

Multi-Column Masking

It is possible to write an algorithm that masks data that depends on other column(s) values. In order to account for the different possible data types, we use an object called a `GenericDataRow` .

Generic Data

A `GenericDataRow` is a map of field names (`String`) to `GenericData` objects. Each `GenericData` object contains the value, along with methods to return the respective typed object. When accessing the value from a `GenericDataObject` it will be necessary to read it into a Core Data Type. To do so, use one of the following methods:

- `getStringValue()`

- `getBigDecimalValue()`

- `getLocalDateTimeValue()`

- `getByteBufferValue()`

Once the value has been masked it should be re-set by calling `setValue` and passing as an argument the value as a Core Data Type.

Batch Masking

By overriding the *maskBatch* method in the `MaskingAlgorithm` interface, an algorithm implementation may increase performance or efficiency in cases where the underlying masking operation may be performed more optimally on multiple values per method call. A common example of this is when the algorithm is accessing an external API to perform masking; in this case, masking multiple inputs per method call allows the access latency of the API to be incurred only once for the entire batch of inputs.

The *maskBatch* method is called with a `MaskingBatch` object parameterized by the same Java type used in the `MaskingAlgorithm` interface definition. The `MaskingBatch` object provides the following methods to facilitate masking:

- *size* - returns the size of the batch of values

- *getValue* - returns the value to be masked at a particular index in the batch

- *setValue* - sets the mask result at a particular index in the batch

- *setError* - indicates that an error occurred when masking the input value at a particular index in the batch

The default implementation of *maskBatch* in the `MaskingAlgorithm` interface provides a simple example of how to use these methods.

> ✏️ **Note**
>
> The masking engine will not utilize the maskBatch method or create a batch with size greater than 1 in all cases. Batch masking is only supported for some job configurations, so it is critical that the mask method also be implemented for all algorithms. It is strongly recommended that the mask and maskBatch method be implemented to produce the same mask results given the same inputs.

Batching is currently supported for these job types:

- All Database masking jobs

- Delimited File masking jobs when no more than one body record type is defined

- Fixed-Width File masking jobs when no more than one body record type is defined

Batch size is equal to the job's `Row Limit` divided by 5, or equal to 2000 when the `Row Limit` is disabled; this is the guaranteed lower bound for batch size, assuming at least that number of inputs are available. Typically, the size of the final batch in a job will be larger.

SDK Workflows

Introduction

This section is intended to walk a developer through several workflows using the Delphix Algorithm SDK, such as creating a new algorithm plugin and installing it on a Delphix Masking Engine. Once an algorithm plugin has been installed, the included algorithms function as expected; they may be assigned to domains and inventory in the normal fashion.

In order to develop and deploy algorithm plugins, you will interact primarily with two tools - the Masking API client, and the Masking Algorithm SDK. The Masking API client is a long-standing feature that allows interactive execution of API operations on the Delphix Masking Engine, while the Masking Algorithm SDK is a new software package created specifically to aid in algorithm development.

Outline for a Guided Tour

By following the steps in the outline below, you can tour the basic functionality provided by the Extensible Algorithm feature and Algorithm SDK.

1. Create an algorithm plugin by choosing one of two options:
   a. Building the sample algorithm project
   b. Creating and building your own algorithm project
2. Run the algorithm plugin using maskApp
3. Install the newly created plugin on the Delphix Masking Engine
4. View and manage the plugins on a Delphix Masking Engine using the API Client
5. Upload multiple plugin in SDK

Building the Sample Plugin

The Algorithm SDK contains a buildable Sample Algorithm Plugin with a number of functional algorithms illustrating the features of the Extensibility Framework. These simple commands build the plugin containing the sample algorithms.

Starting from *sdk_root*:

```
$ cd samples
$ ./gradlew :algorithm:jar
```

This creates the Sample Algorithm plugin JAR file *sdk_root*/samples/build/libs/algorithm.jar.

The Sample Algorithm project provides a convenient way to see a working example plugin.

> 🔥 **Tip**
>
> While it is possible to modify these algorithms by changing the Java source and rebuilding the plugin, when starting a new project to develop one or more standalone algorithms, it is highly recommended that you create your own project rather than modifying files in the Sample Algorithm project subtree. This will prevent the loss of customizations to the project build files should you chose to install a new version of Masking Algorithm SDK over your existing SDK directory.

Creating a New Project

This section describes how to create a brand new Java project for a new masking algorithm plugin. We will use the maskScript utility to create a skeleton project and an empty algorithm class in that project.

Creating the Project

Before you begin, you'll want to pick a name for your project, and an **empty** directory (outside of the Masking SDK source tree) where your project will be created. Once you've done this, run this **maskScript** command:

```
$ maskScript init -t algorithm -d <project path> -n <project name> -a <author name> -v <version>
```

For example, this command will create a project named *demoProject* in the *demo-proj* subdirectory of your home directory.

```
$ maskScript init -d $HOME/demo-proj -n demoProject -a "Demo Author" -v 1.0.0
```

For the rest of this section, we'll assume a new project has been created under *proj_dir*. Change your working directory to *proj_dir*. You'll notice that the project is created with a sample algorithm file *proj_dir*/src/main/java/com/sample/SampleAlgorithm.java. It's possible to build this into a usable plugin by running:

```
$ cd <proj_dir>
$ ./gradlew jar
```

But let's create our own, brand new algorithm.

Creating an Algorithm Class

For this part of the tour, we're going to create a new algorithm named Clobber. First, we'll run the maskScript utility to create a skeleton class file:

```
$ cd <proj_dir>
$ maskScript generate -p com.delphix.demo -c Clobber -v String -s .
```

By convention, the class file Clobber.java will be created under a sub-directory path based on the package name, so it might be he helpful to use the find command to locate it:

```
$ find . -name Clobber.java
./src/main/java/com/delphix/demo/Clobber.java
```

The initial content of this file is:

```
$ cat ./src/main/java/com/delphix/demo/Clobber.java

package com.delphix.demo;

import com.delphix.masking.api.plugin.MaskingAlgorithm;
import java.lang.String;
import javax.annotation.Nullable;

public class Clobber implements MaskingAlgorithm<String> {

    /**
     * Masks String object
     * @param input The String object to be masked. This method should handle null inputs.
     * @return Returns the masked value.
     */
    @Override
    public String mask(@Nullable String input) {
        // TODO: change the default implementation.
            return input;
    }

    /**
     * Get the recommended name of this Algorithm.
     * @return The name of this algorithm
     */
    @Override
    public String getName() {
      // TODO: Change this if you'd like to name your algorithm differently from the Java class.
      return "Clobber";
    }
}
```

Customizing the Algorithm Class

The first thing to notice about the skeleton algorithm is that the mask method just returns the input. This means no masking will be done, so this will certainly need to change. We're going to create an algorithm that overwrites the entire input String with the first letter of that String. This replaces the skeleton *mask* method with:

```
    @Override
    public String mask(@Nullable String input) {
        // Always be ready to handle null or empty input
        if (input == null || input.length() < 2) {
            return input;
        }

        char firstChar = input.charAt(0);
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < input.length(); i++) {
            result.append(firstChar);
        }

        return result.toString();
    }
```

The algorithm name "Clobber" is fine, so we can just delete the TODO comment in the getName() method.

Now, we'll rebuild the project to include this new algorithm in the plugin JAR:

```
$ ./gradlew jar
```

This creates or updates the plugin JAR file *proj_dir*/build/libs/demoProject.jar

Service Discovery

Java service discovery is used to determine which classes in the plugin JAR present relevant functionality to the Delphix Masking Engine. When a plugin is loaded, the file *com.delphix.masking.api.plugin.MaskingComponent* under *META-INF/services* in the JAR is consulted for a list of classes that implement the **MaskingComponent** interface. As **MaskingAlgorithm** includes this interface, each algorithm in the plugin will be discovered this way. In the future, this mechanism may be expanded to support additional types of components beyond algorithms.

> ✏️ **Note**
>
> When the maskScript *generate* sub-command is used to create a new algorithm class, the service discovery metadata file is automatically updated.

If an algorithm class is missing from the services file, it will not be usable when the plugin is loaded. It is essentially invisible to the extensibility framework. If a class is mentioned in this file but not present in the JAR, the plugin will fail to load. There is a fallback during plugin loading that will scan the entire JAR for algorithms if the services file is not present. This fallback may be removed in the future and should not be relied on.

Running an Algorithm Using the SDK Tools

It will often be more convenient to use the SDK utilities to test an algorithm since this avoids the need to install or update your plugin, create masking inventory, and execute jobs on the Delphix Masking Engine. This can be done interactively using maskApp, or entirely from the command line using maskScript.

Using maskApp to Test an Algorithm

The **maskApp** application is interactive and may be launched with no parameters from the shell:

```
$ maskApp
```

After a moment, the application will print a banner and prompt for input. Currently, the only sub-command supported is mask. In order to use this command, you'll need to know the location of the plugin JAR file on the filesystem.

```
MASKING-APP:> mask -j <plugin_jar_location>
```

You will be prompted to select an algorithm. This example runs *maskApp* in **_proj_dir_** and uses the JAR file created with the Create a New Project workflow:

```
MASKING-APP:> mask -j build/libs/demoProject.jar
/Users/******/demo-proj/build/libs/demoProject.jar
Loaded plugin demoProject version 1.0.0 (API version: 1.0.0) from [/Users/******/demo-
proj/build/libs/demoProject.jar]
13:17:00.707 [main] INFO  global - Loaded plugin demoProject: Plugin {'embeddedName': 'demoProject',
'version': '1.0.0', 'author': 'Delphix Dev', 'apiVersion': '1.0.0'}
Framework:
* [0] Clobber
  [1] SampleAlgorithm
Select an algorithm framework: 0
Instance:
* [0] demoProject:Clobber
Select an instance of algorithm framework: Clobber: 0
Selected algorithm: com.delphix.demo.Clobber(Clobber) instance: demoProject:Clobber, data type: STRING
Input value to be masked('null' for null, 'doneMasking' to finish): Test1
Masked value: TTTTT
Input value to be masked('null' for null, 'doneMasking' to finish): test2
Masked value: ttttt
Input value to be masked('null' for null, 'doneMasking' to finish): 1 more test
Masked value: 11111111111
Input value to be masked('null' for null, 'doneMasking' to finish):
```

When you invoke the *mask* sub-command, you will first be presented with a list of possible frameworks (aka. Algorithm Components) to choose from. These correspond with the Java classes that implement the **MaskingAlgorithm** interface in the plugin file. Once you have selected a framework, you will be presented with a list of each pre-defined instance to choose from. If the algorithm supports creating new instances, that option will be present as well. Once an instance is selected, you're ready to enter test values and see the masked result.

> ✏️ **Note**
>
> In order to be usable, each class that implements **MaskingComponent** must also be listed in the appropriate service description file. Refer to this section for details.

The selection marked with a star is the default selection; you may always press return at the prompt to make the default selection.

> ✏️ **Note**
>
> When a Multi-Column algorithm is selected, the prompt will contain the order to provide the input values in. They should be placed on a single row, separated by a comma.
>
> ```
> Enter CSV-formatted values for the following columns in the following order: date1(LOCAL_DATE_TIME),
> date2(LOCAL_DATE_TIME)
> ```

> 🔥 **Missing Algorithms**
>
> If an algorithm seems to be missing from the list, or an algorithm's behavior does not seem to match the latest version of the code, it may be necessary to rebuild the plugin JAR:
>
> ```
> $ cd <proj_dir>; ./gradlew jar
> ```

Running an Algorithm Using maskScript

Algorithms may be also be tested using the SDK **maskScript**. The **maskScript** utility is non-interactive, which lets you conveniently process input files using a masking algorithm. Each input line is considered a separate value to be masked. The algorithm framework and instance are selected using command-line options. This example uses the "Redaction X" algorithm instance from the Sample Algorithm plugin. This plugin can be built using the process described here.

Create a small sample input file:

```
$ echo "Adam
Amy
Brandon" > test_input.txt
```

Mask each line of the file to standard output:

```
$ cat test_input.txt | maskScript mask -j algorithm/build/libs/algorithm.jar -n "Sample Plugin:Redaction Z"
Jun 19, 2020 1:51:54 PM com.delphix.masking.api.provider.LogService info
INFO: Loaded plugin Sample Plugin: Plugin {'embeddedName': 'Sample Plugin', 'version': '1.0.0', 'author':
'Delphix', 'apiVersion': '1.0.0'}
ZZZZ
ZZZ
ZZZZZZZ
Jun 19, 2020 1:51:54 PM com.delphix.masking.api.provider.LogService info
INFO: StringRedaction: Masked a total of 3 values
```

✏️ **Note**

When masking using a Multi-Column algorithm, the inputs in the file must be provided in the same format they would be provided when using the `mask` subcommand of the maskApp (i.e.: comma-separated list of expected values). If unsure of the order, use the `mask` subcommand in the maskApp to see what the expected order is.

🔥 **Redirecting Informational Messages**

To remove all informational message from the output, redirect standard error to /dev/null or an alternate location:

```
$ cat test_input.txt | maskScript mask -j algorithm/build/libs/algorithm.jar -n "Sample Plugin:Redaction Z"
2>/dev/null
```

Installing Multiple Plugins onto the Delphix Masking Engine

Starting SDK version 1.4.0 (corresponding to the Masking Engine version 6.0.8.0) Delphix has implemented multiple plugins upload within SDK, where the Delphix provided dlpx-core plugin is uploaded by default. That gives an option to chain multiple extensible algorithms, even if those are based different plugins.

Load Multiple Algorithm Plugins

1. Using maskApp to Test an Algorithm

   The **maskApp** application is interactive and may be launched with no parameters from the shell:

   ```
   $ maskApp
   ```

   After a moment, the application will print a banner and prompt for input. Currently, the only sub-command supported is mask. In order to use this command, you'll need to know the location of the plugin JAR file on the filesystem.

2. Upload a desired algorithm plugin `bash`

   ```
   MASKING-APP:> mask -j <plugin_jar_location>
   ```

   This example runs *maskApp* in *proj_dir* and uses the JAR file created with the Create a New Project workflow:

   ```
   MASKING-APP:> mask -j ./algorithm/build/libs/plugin1.jar
   /Users/testuser/delphix/masking-algorithm-sdk/./algorithm/build/libs/plugin1.jar
   Loading security manager
   Loaded plugin dlpx-core version 1.4.0 (API version: 1.4.0) from [/Users/testuser/delphix/masking-
   algorithm-sdk/./sdkTools/build/install/maskApp/lib/delphix-algorithm-plugin-1.4.0.jar,
   /Users/testuser/delphix/masking-algorithm-sdk/./algorithm/build/libs/plugin1.jar]
   Loaded plugin plugin1 version 1.4.0 (API version: 1.4.0) from [/Users/testuser/delphix/masking-
   algorithm-sdk/./sdkTools/build/install/maskApp/lib/delphix-algorithm-plugin-1.4.0.jar,
   /Users/testuser/delphix/masking-algorithm-sdk/./algorithm/build/libs/plugin1.jar]
   Framework:
   * [0] dlpx-core:CM Numeric
     [1] dlpx-core:Character Mapping
     [2] dlpx-core:Date Replacement
     [3] dlpx-core:Date Shift
     [4] dlpx-core:Date Shift Discrete
     [5] dlpx-core:Date Shift Variable
     [6] dlpx-core:Dependent Date Shift
     [7] dlpx-core:FullName
     [8] dlpx-core:Name
     [9] dlpx-core:Payment Card
     [10] dlpx-core:Regex Decompose
     [11] dlpx-core:Secure Lookup
     [12] plugin1:Byte Array Redaction
     [13] plugin1:Date Redaction
     [14] plugin1:Number Redaction
     [15] plugin1:Randomized Masking
     [16] plugin1:StringRedaction
   Select an algorithm framework:
   ```

Note: algorithm framework name is now prefexed with the plugin name (that framework is originated from). There are always `dlpx-core` plugin frameworks present, since those are provided by default by the Masking Engine. Previously one could only chain the algorithm with the other algorithm provided by the same plugin. Now it is possible to chain plugin's algorithm with the algorithm instance(s), based on the default dlpx-core plugin.

It is also possible to upload another plugin along with the already loaded ones: instead of selecting an algorithm framework from the above menu - step back by pressing *Ctrl-C*. That will bring you back to the `MASKING-APP:>` menu (with the `UserInterruptException` notification). That error message will be taken care of in a future releases, providing better way for this step back.

```
org.jline.reader.UserInterruptException
Details of the error have been omitted. You can use the stacktrace command to print the full stacktrace.
MASKING-APP:>
```

Here it is possible to use similar `mask -j <plugin_jar_location>` command to upload another plugin:

```
    MASKING-APP:> mask -j ./algorithm/build/libs/plugin2.jar
    /Users/testuser/delphix/masking-algorithm-sdk/./algorithm/build/libs/plugin2.jar
    Loading security manager
    Loaded plugin dlpx-core version 1.4.0 (API version: 1.4.0) from [/Users/testuser/delphix/masking-
algorithm-sdk/./sdkTools/build/install/maskApp/lib/delphix-algorithm-plugin-1.4.0.jar,
/Users/testuser/delphix/masking-algorithm-sdk/./algorithm/build/libs/plugin2.jar]
    Loaded plugin plugin2 version 1.4.0 (API version: 1.4.0) from [/Users/testuser/delphix/masking-
algorithm-sdk/./sdkTools/build/install/maskApp/lib/delphix-algorithm-plugin-1.4.0.jar,
/Users/testuser/delphix/masking-algorithm-sdk/./algorithm/build/libs/plugin2.jar]
    21:17:37.426 [main] INFO  global - Loaded plugin plugin2: Plugin {'embeddedName': 'plugin2', 'version':
'1.4.0', 'author': 'Sample Plugin Author', 'apiVersion': '1.4.0'}
    Framework:
    * [0] dlpx-core:CM Numeric
      [1] dlpx-core:Character Mapping
      [2] dlpx-core:Date Replacement
      [3] dlpx-core:Date Shift
      [4] dlpx-core:Date Shift Discrete
      [5] dlpx-core:Date Shift Variable
      [6] dlpx-core:Dependent Date Shift
      [7] dlpx-core:FullName
      [8] dlpx-core:Name
      [9] dlpx-core:Payment Card
      [10] dlpx-core:Regex Decompose
      [11] dlpx-core:Secure Lookup
      [12] plugin1:Byte Array Redaction
      [13] plugin1:Date Redaction
      [14] plugin1:Number Redaction
      [15] plugin1:Randomized Masking
      [16] plugin1:StringRedaction
      [17] plugin2:MultiColumnDateAlgorithm
      [18] plugin2:Numeric Mapping
      [19] plugin2:RedactionDB
      [20] plugin2:RedactionFile
      [21] plugin2:StringHashedLookup
    Select an algorithm framework:
```

Example above shows 3 plugins uploaded:

- dlpx-core (default)

- plugin1 (uploaded by customer)

- plugin2 (uploaded by customer)

Now it is possible choosing any of those plugins frameworks and their related algorithm instances, as well chaining of those algorithms instances to any configurable extensible algorithm (within the loaded plugins). That behavior simulates Masking Engine where multiple plugins are uploaded.

The technique of algorithms chaining is out of scope of the current description. It's the same as chaining the algorithms belonging to the same plugin. Please refer to the Algorithm Chaining page for chaining details and examples.

Retrieving Information about Installed Plugins

The GET endpoints are useful for getting information about plugins. After following the steps in this section to install the Sample Algorithm plugin, the GET operation will return (elided for brevity):

```
{
  "pluginId": 7,
  "pluginName": "Delphix Sample",
  "originalFileName": "algorithm.jar",
  "originalFileChecksum": "74df61f436aceb80107c22964c027d32a565d0100de36c7fa42f528327cf2e2a",
  "installDate": "2020-06-19T20:15:58.239+0000",
  "installUser": 5,
  "builtIn": false,
  "pluginVersion": "1.0.0",
  "pluginObjects": [
    {
      "objectIdentifier": "2",
      "objectName": "StringRedaction",
      "objectType": "ALGORITHM_FRAMEWORK"
    },
    {
      "objectIdentifier": "3",
      "objectName": "RedactionFile",
      "objectType": "ALGORITHM_FRAMEWORK"
    },
    {
      "objectIdentifier": "5",
      "objectName": "StringHashedLookup",
      "objectType": "ALGORITHM_FRAMEWORK"
    },
    {
      "objectIdentifier": "7",
      "objectName": "Randomized Masking",
      "objectType": "ALGORITHM_FRAMEWORK"
    },
    {
      "objectIdentifier": "Delphix Sample:Byte Array Redaction",
      "objectName": "Delphix Sample:Byte Array Redaction",
      "objectType": "ALGORITHM"
    },
    {
      "objectIdentifier": "Delphix Sample:Date Redaction",
      "objectName": "Delphix Sample:Date Redaction",
      "objectType": "ALGORITHM"
    },
    {
      "objectIdentifier": "Delphix Sample:Number Redaction",
      "objectName": "Delphix Sample:Number Redaction",
      "objectType": "ALGORITHM"
    },
    {
      "objectIdentifier": "Delphix Sample:Numeric Mapping",
      "objectName": "Delphix Sample:Numeric Mapping",
      "objectType": "ALGORITHM"
    },
    ...
  ]
}
```

For each plugin, the plugin metadata, including `pluginId` , `pluginName` and `originalFileChecksum` are displayed first. This is followed by a list of algorithm frameworks included in the plugin, then a list of algorithm instances included in the plugin. The list of frameworks will contain only those frameworks that support the creation of additional algorithm instances as described in this section.

Configurability

Introduction

The Extensible Algorithms feature supports the creation of algorithm frameworks. When an algorithm class is constructed to be a framework, the Delphix Masking Engine operator may create additional instances of the algorithm - with different configurations - after the plugin has been installed. New instances are created by supplying a JSON document describing a new instance using the POST method of the Algorithm endpoint in the Masking Web API. This may be done using the Masking API client. The JSON schema for configuration is determined by which data members in the framework class are marked as configurable, and may vary from framework to framework.

New algorithms created using the SDK skeleton generator are not, by default, configurable using this mechanism. It is necessary to modify the default implementation of the *allowFurtherInstances* method and mark one or more public data members as configurable. This is described in detail in the next section.

This part of the documentation illustrates what options are available when creating an algorithm to define whether and what kind of configuration is required, and what, if any, default instances should be created. It will also describe how to create instances of a plugin provided algorithm framework using the Masking API Client.

Making an Algorithm Configurable

As described in the introduction, there are a couple of requirements for making an algorithm configurable, so that it will appear as a framework on the Delphix Masking Engine:

1. The **getAllowFurtherInstances** method in the algorithm Class must return *true*.

2. One or more data members in the algorithm class must be marked *public*, and must be annotated with the **@JsonProperty** (specifically *com.fasterxml.jackson.annotation.JsonProperty*) annotation.

In order to assure that JSON document and schema interpretation is consistent, most JSON handling is done by the Masking Plugin API implementation, rather than the plugins themselves. For each configurable algorithm, the SDK or Delphix Masking Engine will examine the annotations in the class to determine which values are configurable. Whenever a new instance is created, an attempt is made to apply the user-supplied JSON to the object of the framework class. This includes *some* validation that the supplied JSON matches the expected schema implied by the set of fields marked configurable, however there are some limitations to this validation, as described below.

Example Configurable Algorithm Explained

The concept of configurability can be illustrated using one of the sample algorithms from the SDK as an example - StringRedaction.java in this case:

```java
package sample.masking.algorithm.redaction;
...

public class StringRedaction implements MaskingAlgorithm<String> {
    private String name = "StringRedaction";

    @JsonProperty(value = "redactionCharacter", required = true)
    public String redactionCharacter = "specified";

    @Override
    public String getName() {
        return name;
    }

    @Override
    public Collection<MaskingComponent> getDefaultInstances() {
        StringRedaction instanceX = new StringRedaction();
        instanceX.name = "Redaction X";
        instanceX.redactionCharacter = "X";
        return Arrays.asList(instanceX);
    }

    @Override
    public boolean getAllowFurtherInstances() {
        return true;
    }

    @Override
    public String getDescription() {
        return String.format(
                "Redact String by overwriting with '%s' character", redactionCharacter);
    }

    @Override
    public String mask(@Nullable String input) throws MaskingException {
        if (input == null) {
            return null;
        }
        StringBuilder returnVal = new StringBuilder();

        for (int i = 0; i < input.length(); i++) {
            returnVal.append(redactionCharacter);
        }
        return returnVal.toString();
    }

    @Override
    public void validate() throws ComponentConfigurationException {
        if (redactionCharacter == null || redactionCharacter.length() != 1) {
            throw new ComponentConfigurationException(
                    "redactionCharacter must be a single character");
        }
    }
}
```

This algorithm does simple redaction of the input String, but the redaction character may be configured by creating additional instances with custom values. How this works:

- The Class has a public field redactionCharacter annotated with @JsonProperty. A default value has been provided so that the **getDescription** method will return a suitable description in both the framework and instance cases.

- The Class's **getDefaultInstances** method defines a single instance, with redaction characters 'X'. This is accomplished by simply returning a list of correctly configured objects. The API framework extracts the object configuration as JSON, and store it for use whenever an instance of "Redaction X" algorithm is needed.

- The Class's **getAllowFurtherInstances** method returns true, making it possible to create additional instances of this algorithm after the plugin is loaded on the Masking Engine using the Masking API via the API client.

- The Class implements a **validate** method to ensure that the supplied configuration value is usable. In this case, the length of the redactionCharacter String is restricted to a single character.

Frameworks, Instances, and Configuration Injection

When used as a framework, the algorithm class is instantiated and used without any configuration injection. In the example above, that means that the **getDescription** method will return "Redact String by overwriting with 'specified' character" when the algorithm framework is evaluated. Similarly, **getName** will return "StringRedaction", the name of the framework.

When a runnable algorithm instance is needed, the algorithm class is instantiated, and all saved configuration is injected before any methods are called. This configuration is gathered in one of two ways:

- For statically provided instances embedded in the plugin, the configurable fields of each object returned by the **getDefaultInstances** method are serialized to JSON and saved. Again, only the values of public fields marked with the @JsonProperty annotation are extracted this way.

- When the user creates a new algorithm instance using the Masking Web API, the contents of the *algorithmExtension* field of the POST or PUT request is validated and saved for future injection whenever that particular algorithm instance is needed in the future.

Using the above example again, when algorithm instance "Redaction X" is created, the saved values will be injected, so *redactionCharacter* will have the value 'X'.

Validation of Configuration Values

For what the author can only presume to be performance considerations, the major JSON handling libraries perform only minimal validation when objects are deserialized. The practical effect of this is that several aspects of the @JsonProperty annotation are not enforced. For example, a property might be marked as required, but an object will be successfully deserialized even when that property is missing from the input JSON. While libraries are available that would allow us to expand the degree to which JSON is validated by the framework, this would make defining the exact set of validations done by the API framework vs. what must be validated in the component's validate method even more complex. For these reasons, only minimal input validation is performed by the framework. Plugin authors should validate all aspects of the object's configuration, especially the presence (that is, non-null, non-empty value) of required fields, in the validate method implementation.

However, this is not to say that the unenforced properties of the @JsonProperty annotation should be omitted. These values are visible in the auto-generated schema for each framework, which is visible using the SDK's maskApp, as well as the algorithm/framework endpoint in the Masking API Client, and may be useful for UI generation in the future.

Default Interface Implementations

The Masking Plugin API defines default implementations of **getDefaultInstances** and **getAllowFurtherInstances** as follows:

```
default Collection<ComponentInstanceDescription> getDefaultInstances() {
        return Collections.singletonList(this));
    }

default boolean getAllowFurtherInstances() {
        return getDefaultInstances() == null || getDefaultInstances().isEmpty();
    }
```

This means that if neither of these methods is overridden by the masking algorithm class, a single instance capturing whatever default values exist for configurable fields is created by default.

Only algorithms classes that define **getAllowFurtherInstances** to return *true appear* as Algorithm Frameworks on the Masking Engine.

Build Dependencies for Configurable Algorithms

When the maskScript *init* sub-command is used to create a new project, the initial build files will may not include the dependencies required for the Jackson @JsonProperty annotation. This an be corrected by adding this line to *proj_root*/gradle.properties:

```
jacksonVer=2.9.5
```

And this line to the **dependencies** section at the end of *proj_root*/build.gradle:

```
compileOnly ('com.fasterxml.jackson.core:jackson-annotations:' + jacksonVer)
```

The set of Jackson annotations tested and supported for use in algorithm plugin classes are:

- *@JsonProperty*
- *@JsonPropertyDescription*
- *@JsonFormat* (Useful in specifying formats for Date fields)

Using an Algorithm Framework

When a plugin algorithm supports configuration, it is possible to create new instances of the algorithm on the Delphix Masking Engine by specifying the desired configuration. This is done using the engine's Masking Web API. Configurable algorithms may also be tested using the **maskApp** and **maskScript** utilities, by providing the desired configuration in an input file or at the command line.

Creating New Algorithm Instances Using the maskApp SDK Utility

When the maskApp utility's *mask* command is invoked and a configurable algorithm is selected, the option will be presented to create a new algorithm instance. This is done by choosing "::Create New Instance::". The algorithm's configuration schema is displayed, and then a valid JSON input must be provided to create the new instances. Rather than entering the literal JSON, the '@' symbol may be used to load the JSON from a file (@*file-path*).

What follows is an example of loading the Sample Algorithm plugin, creating a new instance of the *StringRedaction* framework and masking test values with the new algorithm instance.

```
$ maskApp
... Startup Messages ...
MASKING-APP:> mask -j algorithm/build/libs/algorithm.jar
/Users/jleser/ws/algorithm-sdk/algorithm/build/libs/algorithm.jar
Loaded plugin Delphix Sample version 1.0.0 dea904c (API version: 1.0.0) from [/Users/jleser/ws/algorithm-
sdk/algorithm/build/libs/algorithm.jar]
16:44:47.743 [main] INFO  global - Loaded plugin Delphix Sample: Plugin {'embeddedName': 'Delphix Sample',
'version': '1.0.0 dea904c', 'author': 'Delphix', 'apiVersion': '1.0.0'}
Framework:
* [0] Byte Array Redaction
  [1] Date Redaction
  [2] Number Redaction
  [3] Numeric Mapping
  [4] Randomized Masking
  [5] RedactionFile
  [6] StringHashedLookup
  [7] StringRedaction
Select an algorithm framework: 7
Instance:
* [0] Delphix Sample:Redaction X
  [1] Delphix Sample:Redaction Y
  [2] Delphix Sample:Redaction Z
  [3] ::Create New Instance::
Select an instance of algorithm framework: StringRedaction: 3
The JSON schema of the selected framework is:
{
  "type" : "object",
  "id" : "urn:jsonschema:sample:masking:algorithm:redaction:StringRedaction",
  "properties" : {
    "redactionCharacter" : {
      "type" : "string",
      "required" : true
    }
  }
}
Enter config(Prefix with '@' for file location)(Blank for no config): { "redactionCharacter" : "+" }
Enter instance name: RedactPlus
Algorithm Configuration: {"redactionCharacter":"+"}
Selected algorithm: sample.masking.algorithm.redaction.StringRedaction(StringRedaction) instance:
RedactPlus, data type: STRING
Input value to be masked('null' for null, 'doneMasking' to finish): Test
Masked value: ++++
Input value to be masked('null' for null, 'doneMasking' to finish): One
Masked value: +++
Input value to be masked('null' for null, 'doneMasking' to finish): TwoThree
Masked value: ++++++++
```

Creating New Algorithm Instances on the Delphix Masking Engine

New instances of plugin frameworks may be created using the Delphix Masking Engine's Web API's *algorithm*
endpoint. This is similar to creating any other algorithm using the *algorithm* API endpoint and may be performed
using the API client. Unlike when an algorithm is created using older, built-in frameworks like Secure Lookup:

- The value for *algorithmType* in the JSON request is always "COMPONENT". This is now the default value, so this
  field may be omitted.

- A value for the field *frameworkId* must be included - this is the integer ID of the framework as provided in the plugin description retrievable using the GET operation on the plugin endpoint, or GET on the *algorithm/frameworks* endpoint.

- The *algorithmExtension* field's contents are used directly as the JSON configuration for the algorithm instance. Unlike other algorithm types, this field does not have a fixed schema for COMPONENT type algorithms. The required schema may be retrieved using the procedure described below.

This example API request, POSTed to the algorithm endpoint, creates a new instance of the StringRedaction algorithm (described above), named "RedactStar" using '*' as the redaction character. In this case, the sample algorithm plugin JAR has already been uploaded, and the StringRedaction framework has id 19:

```
{
  "algorithmName": "RedactStar",
  "algorithmType": "COMPONENT",
  "description": "Redact with the star character",
  "frameworkId" : 19,
  "algorithmExtension" : {
    "redactionCharacter": "*"
  }
}
```

Discovering the algorithmExtension API Field Schema

The Masking Web API *algorithm/framework* endpoint has the ability to show the JSON Schema for each algorithm framework implemented using the extensibility mechanism. By default, the schema is not included, but by setting *include_schema* true, the schema may be retrieved. Here is the GET API result, including schema, for the StringRedaction framework used above:

```
{
  "frameworkId": 19,
  "frameworkName": "StringRedaction",
  "frameworkType": "STRING",
  "plugin": {
    "pluginId": 47,
    "pluginName": "algorithm"
  },
  "extensionSchema": {
    "id": "urn:jsonschema:sample:masking:algorithm:redaction:StringRedaction",
    "properties": {
      "redactionCharacter": {
        "type": "string",
        "required": true
      }
    }
  }
}
```

This schema is generated automatically using the annotated public fields in the framework class.

Using Multi-Column Algorithms

To be able to configure and use the Multi-Column (MC) Algorithms one should be familiar with the following themes:

- Extensible Algorithms in general

- Their creation using the Masking SDK

- Extensible Algorithms Plugin installation

- Masking API Client (optional)

Logical Fields

A sample instance (serving as an example) of the MC algorithms is in the Masking SDK distribution, named "MultiColumnDateAlgorithm". That framework (the instance is based on) defines two fields:

```
    @Override
    public Map<String, MaskingType> listMaskedFields() {
        /*
         *  Here we define the column names to be used in the algorithm. These names are only used to
reference the
         *  columns within the algorithm and do not need to correspond to the names of the columns on the
data source.
         *  For example, our data source may call these 2 fields "dateOfBirth" and "dateOfDeath", however
within the
         *  algorithm implementation they will be referenced as "startDate" and "endDate" (see mask method
to see how
         *  this is used).
         */
        Map<String, MaskingType> maskedFields = new HashMap<String, MaskingType>();
        maskedFields.put("startDate", MaskingType.LOCAL_DATE_TIME);
        maskedFields.put("endDate", MaskingType.LOCAL_DATE_TIME);
        return maskedFields;
    }
```

In that example, the fields `"startDate"` and `"endDate"` are logical fields, defined by the framework. If one doesn't have access to the source code of the framework, it's possible to find the logical names (and where types) using the *Masking API: GET /algorithms/{algorithmName}* endpoint.

Let's suppose you already have an instance of MC Algorithm installed on the ME. That might happen in any of the following two cases:

- The Plugin you've installed contains a default instance for MC algorithms.

- The Plugin you've installed contains only a framework for configurable MC algorithms. In that case, you've configured an instance of the algorithm.

Let's take as an example the above mentioned "MultiColumnDateAlgorithm" algorithm (plugin is named "sample" in that example). Retrieving its info using the *GET /algorithms/{algorithmName}* endpoint returns:

Response Body

```json
{
    "algorithmName": "sample:MultiColumnDateAlgorithm",
    "algorithmType": "COMPONENT",
    "pluginId": 37,
    "fields": [
        {
            "fieldId": 4,
            "name": "endDate",
            "type": "LOCAL_DATE_TIME"
        },
        {
            "fieldId": 5,
            "name": "startDate",
            "type": "LOCAL_DATE_TIME"
        }
    ],
    "algorithmExtension": {}
}
```

Here we can see the {fieldId, name, and type} information structure for the logical fields, defined by the current framework. We will use that data when configuring the Inventory fields.

Configuring columnMetadata for MC algorithm

To configure the involved column (i.e. masked and read-only columns) - we should update the column's metadata with the following information:

```
"algorithmFieldId"
"algorithmGroupNo"
"algorithmName"
"domainName"
```

The last two fields are the regular configuring fields for masked columns. Let's look closer to the newly introduced fields for MC:

- `algorithmFieldId` is a filedId for the corresponding logical field. For example for "startDate" from the example above its value is 5.

- `algorithmGroupNo` is a group number (integer) for the columns treated by the same algorithm instance. It is introduced for cases where we might have multiple columns of a similar type, which are masked by the different Masking Jobs using the same algorithm. In such a case that's important to unite the columns per algorithm run, by assigning the same group number.

There are two supported methods to configure the columnMetadata for the masked table inventory:

- Via API

- Via UI

Configuring columnMetadata for MC algorithms via API

Below is the example of the column metadata before it's configured for MC algorithm:

Response Body

```
{
    "columnMetadataId": 63,
    "columnName": "DATA00",
    "tableMetadataId": 19,
    "dataType": "VARCHAR2",
    "columnLength": 100,
    "isMasked": false,
    "isProfilerWritable": true,
    "isPrimaryKey": false,
    "isIndex": false,
    "isForeignKey": false
},
```

Let's associate that field with the logical field `startDate` (fieldId=5) from the snapshot above, by adding the mentioned fields:

Response Body

```
{
    "columnMetadataId": 63,
    "columnName": "DATA00",
    "tableMetadataId": 19,
    "algorithmName": "sample:MultiColumnDateAlgorithm",
    "algorithmFieldId": 5,
    "algorithmGroupNo": 1,
    "domainName": "DOB",
    "dataType": "VARCHAR2",
    "dateFormat": "yyyy-MM-dd",
    "columnLength": 100,
    "isMasked": true,
    "isProfilerWritable": true,
    "isPrimaryKey": false,
    "isIndex": false,
    "isForeignKey": false,
    "notes": ""
},
```

> ✏️ **INFO**
>
> For the masked column, the *isMasked* field should be manually changed to *true*, while for read-only field it stays *false*.

If at this point an inventory for the masked table is checked in the UI - the configured (via API) inventory will be displayed there:

Configuring columnMetadata for MC algorithms via UI

The same columnMetadata configuration can also be made via the UI. As with other algorithms one has to choose the Domain and Algorithm values, applied to the current column. If a Multi-Column algorithm has been chosen, the following additional two fields will need to be filled out:

- `Select Logical Field` dropbox, where the corresponding logical field to be selected.
- `Algorithm Group` window, where algorithmGroupNo value to be entered.

Edit Properties

Column Name

DATA00

Notes

ID Method

Auto ▼

Domain

DOB ▼

Algorithm

sample:MultiColumnDateAl… ▼

Select Logical Field

startDate ▼

Algorithm Group

1

Type: LOCAL_DATE_TIME

Date Format

yyyy-MM-dd ▼ ❓

Cancel   Save

✏️ **INFO**

In the UI configuration for columnMetadata, the customer shouldn't mark the *isMasked* field (as via the API in the example above). It's taken care automatically since ME knows the associated logical field is being masked or used as a read-only.

Error Management

There are different configuration errors possible while setting the MC algorithms. The configuration process prevents as many misconfigrations as possible, but some configuration errors can only be detected when a job is executed. For example, if trying to associate a second column to the same (already busy) logical field will result in a configuration error similar to:

Error Properties
The algorithm field 'startDate' {id:"5", seq: "1"} is already assigned to field 'DATA00' {id:"63"}.
Column Name   Notes

In case there is a missed association with the required logical field - that type of error isn't recognized during the configuration, but only during the job execution (which will fail due to that misconfiguration).

Please find below an example of the monitor job error report:

⚠ Error Report   ✕

Execution Events			❔ Learn More
Event	**Cause**	**Description**	
JOB_ABORTED	UNHANDLED_EXCEPTION	Exception:Algorithm 'sample:MultiColumnDateAlgorithm' requires 2 fields [endDate, startDate] but found 1 fields [startDate]. Missing fields: [endDate]	

75_53.log

2020-12-14 17:24:16,233 [thread] INFO com.dmsuite.dmsApplicator.masking.XMLGenerator executeMarshalling - Generate request xml started successfully.
2020-12-14 17:24:16,782 [thread] INFO com.dmsuite.dmsApplicator.masking.XMLGenerator executeMarshalling - Generate Request xml done successfully.
2020-12-14 17:24:16,810 [thread] INFO com.dmsuite.dmsApplicator.masking.transformation.MaskingMarshalling createKettleXML - Generate Transformation XML started successfully

Limitations for the MC Algorithms

1. Currently, it's possible to run the MC Algorithms only on a single table. Masking multiple tables columns by MC Algorithms is not supported.

2. XML File masking does not support MC algorithms.

3. VSAM File masking does not support MC algorithms. The only exception is VSAM files which don't redefine record types.

4. Some types of misconfiguration errors (as described above) are only detected during job execution.

Service Interfaces

Introduction

The Extensible Algorithms framework makes a number of services available to the algorithm implementation. This prevents the algorithm from having to re-implement code to perform certain routine tasks and facilitates seamless integration with the Masking Engine. This functionality is exposed to the algorithm class via the **ComponentService** interface.

Whenever a new Masking Algorithm instance is required for masking, the extensibility framework first injects any saved configuration, then invokes the objects *setup* method. This method is passed a reference to an object that implements **ComponentService**. The algorithm's *setup* method can then use this object to access a number of provider methods:

- *getInstanceName* - Get the name of this instance. Because the instance name it is not typically a configurable field in the algorithm, the *getName* method will not correctly return the name of an algorithm instance, even after JSON configuration injection. This method will always return the correct instance name as known to the Masking Engine.
- *openInputFile* - Access the contents of a file, as described in this section
- *getAlgorithmByName* - Get a usable instance of another algorithm, as described in this section
- *getCryptoService* - Access cryptographic methods based on the algorithm's key, as described in this section
- *getLogService* - Get a logger object, as described in this section

> 🔥 **Getting More Information**
>
> Refer to the *com.delphix.masking.api.provider* package in the Javadoc for detailed information.

Accessing Files

It is often the case that a masking algorithm will require a large library of input values - for example, a set of replacement names or account numbers. In other cases, it may be desirable to store the configuration of a particularly complex algorithm in a format other than JSON, perhaps in order to leverage pre-existing code. To support these use cases, the extensible algorithm framework allows algorithms to access input files from a variety of sources.

Opening Input Files

Algorithms may access files in several locations, both on the engine and over the network. In all cases, access is achieved by invoking the *openInputFile* method of the **ComponentService** object passed to the algorithm's *setup* method to acquire an **InputStream**. The file's location must be specified by an **FileReference** object visible in the object's public fields. This object is passed to the *openInputFile* method. The value of the **FileReference** may be made configurable using the @JsonProperty annotation. The *openInputFile* method accepts a URI style syntax that combines standard URL notation for web resources with custom URI types for engine and JAR located files. The following formats are supported for FileReference values:

URI Format	Description
`http[s]://<host>[:<port>]/<path>`	To open files located on a remote web server
`jar://file/<filepath>`	To open a file located in the algorithm plugin JAR
`delphix-file://upload/<file reference details>`	To open a file uploaded using Delphix Masking Engine's *fileUpload* endpoint. The result of **POST>**ing to the *fileUpload* API endpoint is a URI in this format that should be used exactly as-is for the *uri* value of the **FileReference**.
`delphix-file://mount/<mountType>/<mountId>/<file path>`	To open a file located on a NFS/CIFS mount server that has been mounted inside the Delphix Masking Engine using mountFilesystem endpoint

Example Algorithm

```
public class RedactionFile implements MaskingAlgorithm<String> {
    private String redactionCharacter = null;

    @JsonProperty(value = "file", required = true)
    public FileReference file;
    @Override
    public String getName() {
        return "RedactionFile";
    }

    @Override
    public String mask(@Nullable String input) throws MaskingException {
        if (input == null) {
            return null;
        }
        StringBuilder returnVal = new StringBuilder();
        for (int i = 0; i < input.length(); i++) {
            returnVal.append(redactionCharacter);
        }
        return returnVal.toString();
    }

    @Override
    public void setup(@Nonnull ComponentService serviceProvider) {
        InputStream inputStream = serviceProvider.openInputFile(file);
        try (Scanner scanner = new Scanner(inputStream, Charset.defaultCharset().name())) {
            redactionCharacter = scanner.nextLine().trim();
        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException("Unable to parse input file", e);
        }
    }

    @Override
    public void validate() throws ComponentConfigurationException {
        GenericReference.checkRequiredReference(file, "file");
    }
}
```

Some methods have been omitted for brevity.

This example algorithm is very similar to the StringRedaction class discussed earlier, in that it redacts strings by replacing with a same-length string of the redaction character. This variant reads the character to use for redaction from an input file, the location of which is specified in the algorithm's configuration. This is all done in the initialize method:

- The value of the variable file is public and marked configurable with @JsonProperty.

- The file reference is passed to serviceProvider.*openInputFile* during setup - this allows the algorithm to ingest input files in any supported location.

- The redaction character is read from the input stream and stored in instance variable redactionCharacter for use in the *mask* method.

- This class's validate method uses the static method **GenericReference**.*checkRequiredReference* provided by the Masking Plugin API to check the file reference for validity.

Accessing Database Servers (JDBC)

It is often the case that a masking algorithm will require access to a large amount of data such as lookup values for masking input data or storing states of the algorithm. To support these use cases, the extensible algorithm framework allows algorithms to access database servers using JDBC connections.

Opening Database Connection

Algorithms access the database by using extensible drivers. Access is achieved by invoking the *openJdbcConnection* method of the **ComponentService** object passed to the algorithm's *setup* method to acquire a **Connection (java.sql.Connection)** object. The file's location must be specified by an **JdbcReference** object visible in the object's public fields. This object is passed to the *openJdbcConnection* method. The value of the **JdbcReference** may be made configurable using the @JsonProperty annotation. The **JdbcReference** object requires following fields

1. *jdbcDriverId*: The driver Id of the JDBC Driver uploaded using the JDBC Driver API (*/jdbc-drivers*).

2. *url*: The JDBC URL that will be used to connect to the server. Please don't use this to pass the credentials.

3. *credFileReference*: A **FileReference** object that contains the location of the JSON file that stores the credentials. The schema of the file is:

```
{
  "username": "USERNAME",
  "password": "PASSWORD"
}
```

The file must be a *mount* type **FileReference** object. To see how to create a mount type **FileReference** object, refer to Accessing Files. The **Connection** object is kept open throughout the execution of the algorithm unless it is closed by the algorithm itself.

Example Algorithm

```java
public class RedactionDB implements MaskingAlgorithm<String> {
    private String redactionCharacter = null;

    @JsonProperty(value = "jdbc", required = true)
    @JsonPropertyDescription("A reference to a database containing a table redaction_character")
    public JdbcReference jdbc;

    private static final String GET_REDACTION_CHARACTER = "SELECT redact FROM redaction_character LIMIT 1";


    @Override
    public String getName() {
        return "RedactionDB";
    }

    @Override
    public String mask(@Nullable String input) throws MaskingException {
        if (input == null) {
            return null;
        }
        StringBuilder returnVal = new StringBuilder();
        for (int i = 0; i < input.length(); i++) {
            returnVal.append(redactionCharacter);
        }
        return returnVal.toString();
    }

    @Override
    public void setup(@Nonnull ComponentService serviceProvider) {
      try (Connection conn = serviceProvider.openJdbcConnection(jdbc);
            PreparedStatement stmt = conn.prepareStatement(GET_REDACTION_CHARACTER)) {
            ResultSet resultSet = stmt.executeQuery();
            List<String> redactionChars = new ArrayList<>();
            if (resultSet.next()) {
              redactionCharacter = resultSet.getString("redact");
            } else {
              throw new RuntimeException("Couldn't find redaction character");
            }
      } catch (SQLException e) {
            throw new RuntimeException(e);
      }

    }

    @Override
    public void validate() throws ComponentConfigurationException {
        GenericReference.checkRequiredReference(jdbc, "jdbc");
    }
}
```

Some methods have been omitted for brevity.

This example algorithm is very similar to the StringRedaction class discussed earlier, in that it redacts strings by
replacing with a same-length string of the redaction character. This variant reads the character to use for redaction
from a table in a database, the connection information for which is specified in the algorithm's configuration. This is
all done in the initialize method:

- The value of the variable "jdbc" is public and marked configurable with @JsonProperty.

- The jdbc reference is passed to serviceProvider.*openJdbcConnection* during setup.

- The redaction character is read from the table *redaction_character* and stored in the instance variable *redactionCharacter* for use in the *mask* method.

- This class's validate method uses the static method **GenericReference**.*checkRequiredReference* provided by the Masking Plugin API to check the jdbc reference for validity.

Algorithm Chaining

The extensible algorithm framework allows algorithms to instantiate and call other algorithms. This is useful to allow for the composition and reuse of algorithm behaviors. This feature is referred to as algorithm chaining.

Calling Other Algorithms

In order to make use of this feature, the caller algorithm must acquire an object of the algorithm class it wishes to call by requesting it by instance name using the *getAlgorithmByName* method of the **ComponentService** object. This is done during the execution of the algorithm's *setup* method.

This method requires that the caller specify two values:

1. A reference to the algorithm instance. This must be stored in an **AlgorithmInstanceReference** object whose value is the name of the algorithm instance. This is *algorithmName* in the Masking API, occasionally referred to as "algorithmCd" or "algorithm code". The **AlgorithmInstanceReference** object must be referenced in a *public* field in the algorithm object.

2. The type of data the returned algorithm object should mask, selected from the core types supported by the extensible algorithm framework. Type adaptation is not currently supported in this context, so the algorithm's native type must be the type requested using *getAlgorithmByName*.

Once an algorithm object has been obtained using *getAlgorithmByName*, a reference to the algorithm object maybe kept and that algorithm's *mask* method called as needed.

Examples:

- You are creating algorithm instance A via the Masking API Client Algorithm endpoint, and algorithm A uses *getAlgorithmByName* to find algorithm B during *setup*. For the creation of algorithm A to succeed, algorithm B **must** already exist on the Delphix Masking Engine.

- You are installing a plugin that would create the same algorithm A as a static instance. This will fail if algorithm instance B is not also provided by an algorithm class in the same plugin.

Because it is difficult to predict what algorithm names exist on a Delphix Masking Engine, it is advised that the names of any algorithms used for chaining be supplied in the algorithm's JSON configuration. Hard-coding names of algorithms passed to *getAlgorithmByName* directly in the Java source creates dependencies that are not visible except in the error message that results when the caller algorithm fails to initialize, as described in the second example scenario above. Hard-coded references to other algorithms provided by the same plugin should have the value ":**algorithmName**". The ":" character tells the API to fill in this plugin's name when searching for the instance.

> ✏️ **Note**
>
> Algorithm instances provided by plugins (via the *getDefaultIntances* method) are prohibited from having dependencies on algorithm instances provided by other plugins. A way to safely implemented this kind of dependency may be added in the future.

Example Algorithm

```java
public class RandomizedStringMasking implements MaskingAlgorithm<String> {
    private List<MaskingAlgorithm<String>> algorithmList = new ArrayList<>();
    private Iterator<Integer> randomStream;

    @JsonProperty(value = "algorithmNames", required = true)
    public List<AlgorithmInstanceReference> algorithms;

    @Override
    public String getName() {
        return "Randomized Masking";
    }
    @Override
    public Collection<MaskingComponent> getDefaultInstances() {
        RandomizedStringMasking myInstance =
                new RandomizedStringMasking() {
                    @Override
                    public String getName() {
                        return "Randomized Redaction";
                    }
                    @Override
                    public String getDescription() {
                        return "Apply a random redaction algorithm from { X, Y, Z }";
                    }
                };
        myInstance.algorithms =
                Arrays.asList(
                        new AlgorithmInstanceReference(":Redaction X"),
                        new AlgorithmInstanceReference(":Redaction Y"),
                        new AlgorithmInstanceReference(":Redaction Z"));

        return Collections.singletonList(myInstance);
    }

    @Override
    public void validate() throws ComponentConfigurationException {
        if (algorithms == null || algorithms.isEmpty()) {
            throw new ComponentConfigurationException(
                    "Value for field algorithmNames is missing or empty");
        }
        for (AlgorithmInstanceReference ref : algorithms) {
            GenericReference.checkRequiredReference(ref, "algorithms");
        }
    }

    @Override
    public void setup(@Nonnull ComponentService serviceProvider) {
        for (AlgorithmInstanceReference algorithm : algorithms) {
            algorithmList.add(serviceProvider.getAlgorithmByName(algorithm, MaskingType.STRING));
        }
        randomStream = new Random().ints(0, algorithmList.size()).iterator();
    }

    @Override
    public String mask(@Nullable String s) throws MaskingException {
        return algorithmList.get(randomStream.next()).mask(s);
    }
```

Some methods have been omitted for brevity.

This algorithm is configured with a list of other String masking algorithms and masks by calling another algorithm from that list at random. This randomization is not based on the algorithm key, so results will not be consistent across masking runs. In addition, this framework defines a default instance that chooses randomly between algorithms "Redaction X", "Redaction Y" or "Redaction Z" included in the same plugin.

The algorithm's public fields include a list of **AlgorithmInstanceReference** objects, made configurable by the JsonProperty annotation.

This algorithm's *setup* method does the following:

- For each algorithm name, it calls *getAlgorithmByName* to instantiate a usable algorithm object, saving them in *algorithmList*.
- It initializes a random number generator to produce integers corresponding to each index in *algorithmList*.

This algorithm's *mask* method selects an algorithm at random from *algorithmList* and calls it's *mask* method on the input value, returning the result.

This algorithm's *getDefaultInstances* method creates a single instance that chooses between three algorithms. Each algorithm reference begins with ':', indicating that these algorithms should be found in the same plugin as this algorithm. The getName and getDescription methods of the returned object are overridden to provide values different from those of the framework itself.

Using Cryptographic Keys

Cryptography is useful in algorithm development for a range of purposes, from straightforward encryption of value to shuffling collections and permuting data in a manner that is consistent across masking jobs. The extensible algorithm framework automatically provides each algorithm with a cryptographic key. This key is wrapped by a service provider object that implements the **CryptoService** interface, providing a number of useful operations based on the algorithm's key. It is also possible to retrieve the raw key assigned to the algorithm as an array of bytes.

Similar to working with files, there is a **KeyReference** type that represents a reference to the key. This is present to support access to keys stored in alternative locations (ex. a key vault) in the future. Currently, the only supported value for these references is "", which indicates that the per-algorithm key stored on the Delphix Masking Engine should be used.

> ✏️ **Note**
>
> When working with the Masking SDK maskApp and maskScript utilities, each algorithm's key is a stable hash of its algorithm name, but maybe temporarily set to a random value using the -K flag.

Using the CryptoService Provider

The first step any algorithm that wishes to use its algorithm key must take is to retrieve a handle to a cryptographic service provider during initialization. This is done by calling the **ComponentService** object's *getCryptoService* method. The returned provider wraps the key. The operations supported by the **CryptoService** interface are as follows:

- *getRawKey* - retrieve the raw key associated with this provider as a byte array.

- *wrap* - wraps an array of bytes to create a CryptoService object. This is useful for accessing CryptoService methods when the algorithm's key is stored in an alternative location or hard-coded in the algorithm source. This is not recommended, but potentially useful to support use cases in existing custom algorithms.

- *deriveNewKey* - derive a new key by permuting this provider's key using SHA-256. A new CryptoService object is returned wrapping the new key. The zero-argument version of this method returns the same key each time it is called on the same provider - in order to create multiple, different keys, a different salt must be provided to each method call. It is advisable that whenever an algorithm wishes to use cryptography for multiple purposes, new and distinct keys be derived for each purpose.

- *computeHashedLookupIndex* - compute an integer value from 0 to (modulus - 1) by hashing the input value + key. This method is designed to allow randomized, but consistent, lookups into a replacement table based on the input value.

- *shuffleList* and *shuffleListNoCollisions* - these methods shuffle their argument **List** in-place using the key to seed the randomization. The "noCollisions" variant ensures that no object in the list remains in its original position.

Example Algorithm

```
public class StringHashedLookup implements MaskingAlgorithm<String> {
    private List<String> replacements;
    private CryptoService crypto;

    public KeyReference key = new KeyReference();

    @JsonProperty("replacementFile")
    public FileReference replacementFile;

    @Override
    public void validate() throws ComponentConfigurationException {
        GenericReference.checkRequiredReference(replacementFile, "replacementFile");
    }

    @Override
    public void setup(@Nonnull ComponentService serviceProvider) {
        replacements = new ArrayList<>();

        String line;
        try (InputStream is = serviceProvider.openInputFile(replacementFile);
             BufferedReader reader =
                 new BufferedReader(new InputStreamReader(is, "UTF_8"))) {
            while ((line = reader.readLine()) != null) {
                replacements.add(line);
            }
        } catch (IOException e) {
                throw new RuntimeException(e);
        }
        crypto = serviceProvider.getCryptoService(key);
    }

    @Override
    public String mask(@Nullable String input) {
        if (input == null || input.length() == 0) {
            return input;
        }
        return replacements.get((int) crypto.computeHashedLookupIndex(input, replacements.size()));
    }
}
```

Some methods have been omitted for brevity.

This example algorithm functions very similarly to the existing Secure Lookup algorithm, except it employs a different
hash method from the new **CryptoService** provider.

- The algorithm is configured with an input file by supplying a public, annotated **FileReference** field
  replacementFile.

- In the *setup* method, the replacement file is ingested and saved as a list of values.

- Additionally in *setup*, the cryptographic service provider is initialized using the default key reference, accessing the
  algorithm's key.

- The mask method uses the *computeHashedLookupIndex* method to compute the index of the replacement to use
  from the *replacements* list.

Logging

It is possible for a plugin algorithm to write information into the job logs, and consequently, Delphix Masking Engine logs. This is accomplished by using calling the *getLogService* method of the **ServiceProvider** interface provided at the algorithm setup. The resulting **LogService** object may be used to make logging entries at various levels of severity. The available log levels are ERROR, WARNING, INFO, and DEBUG.

The log interface is provided to allow for debugging output during algorithm development, and for reporting of statistical or similar values detailing the overall operation of the algorithm, typically in the *tearDown* method.

> 🔥 **Logging Security Warning**
>
> An algorithm **must** never log unmasked values (the *input* argument to the *mask* method) to the log files. The job and Masking Engine log files may be retrieved by engine users and are included support bundles.

> 🔥 **Logging Verbosity**
>
> Algorithms also should not log progress messages or other verbose details, especially from the *mask* method, as this will fill the log files with messages and may impact job performance. There is a rate-limiting mechanism that limits the volume of messages each algorithm can write over time, but any amount of routine logging is likely to diminish the overall usefulness of the logs by obscuring more important messages.

Example Code

This example is take from the StringRedaction sample algorithm provided with the SDK:

```java
public class StringRedaction implements MaskingAlgorithm<String> {
    ...
    private LogService logger;
    ....

    @Override
    public String mask(@Nullable String input) throws MaskingException {
        if (input == null) {
            return null;
        }

        if (random.nextDouble() < 0.1) {
            logger.info("{0}: Masked {1} values", getName(), count);
        }

        StringBuilder returnVal = new StringBuilder();

        for (int i = 0; i < input.length(); i++) {
            returnVal.append(redactionCharacter);
        }
        count++;
        return returnVal.toString();
    }

    @Override
    public void validate() throws ComponentConfigurationException {
        if (redactionCharacter == null || redactionCharacter.length() != 1) {
            throw new ComponentConfigurationException(
                    "redactionCharacter must be a single character");
        }
    }

    @Override
    public void setup(@Nonnull ComponentService serviceProvider) {
        logger = serviceProvider.getLogService();
    }

    @Override
    public void tearDown() {
        logger.info("{0}: Masked a total of {1} values", getName(), count);
        count = 0;
    }
}
```

Some methods and fields elided for the sake of brevity

The relevant details here:

- The *setup* method uses the provided **ComponentService** object to get a **LogService** instance, saving it as *logger*.

- The *mask* method calls the logger's *info* method to write informational messages at random during execution. This kind of "progress" logging may be useful during development but should be removed for algorithms before production deployment.

- The *tearDown* method calls the *info* method of the logger again to record the total number of values masked.

Security Considerations

Introduction

It important that only well-crafted and trustworthy plugin modules are installed on the Delphix Masking Engine; otherwise, the security of the appliance and masked data may be compromised. This section contains information for developers on how to ensure that their algorithm plugins function securely, as well as for engine administrators to ensure that only trusted plugins are installed and executed on the engine.

Algorithm Implementation

This section details a number of security consideration developers should be aware of when creating plugin algorithms for the Delphix Masking Engine.

The Security Sandbox

During execution, all plugin code is sandboxed using the Java Security Manager. Plugins are granted all permissions *except* for the following non-`FilePermission`:

Class	Target	Action
java.net.SocketPermission	localhost:-	accept, connect, listen, resolve
java.lang.RuntimePermission	exitVM	
java.lang.RuntimePermission	createClassLoader	
java.lang.RuntimePermission	accessClassInPackage.sun	
java.lang.RuntimePermission	setSecurityManager	
java.security.SecurityPermission	setPolicy	
java.security.SecurityPermission	setProperty.package.access	

With regards to FilePermissions, `read` access is granted to all, though `write` is only allowed for the following directories:

- the masking user's home directory (`System.getProperty("user.home")`)
- the JVM's default temp directory (`System.getProperty("java.io.tmpdir")`)

Please note that both of these locations are shared, so care will need to be taken to avoid collisions.

The set of permissions granted to plugins is static and cannot be modified. To facilitate testing, the same security restrictions are applied when plugins are run using the *maskApp* or *maskScript* utilities in the Masking SDK (with the exception of the `SocketPermission` and all instances of `write` `FilePermission`).

Handling Errors

One important aspect of ensuring that an algorithm securely masks sensitive data is proper handling any errors that might occur during algorithm execution.

One particular category of error that might occur is when the input value does not match the format expected by the algorithm. Perhaps an account number masking algorithm is applied to a column containing free-text comments, or an image blurring algorithm is applied to non-image binary data. This is referred to as Non-conformant data. The Algorithm Extension Plugin API defines how an algorithm may trigger the Non-conformant data handling mechanisms built into the Masking Engine.

Reporting Non-conformant Data

Whenever a Non-conformant input value is encountered, and the algorithm cannot mask it, the algorithm *mask* method should throw an exception of class **NonConformantDataException** supplied by the Masking Plugin API. This triggers the Non-conformant data reporting mechanism of the masking engine. The **String** value used to construct this exception **must not** include the unmasked input value, as this would result in the sensitive value being saved in the Masking Engine logs and made visible in the engine UI. A redacted sample of the Non-conformant data will be saved automatically by the reporting mechanism.

Handling Other Errors

In general, other code errors should be handled as responsibly as possible by the algorithm implementations, following these guidelines:

- Under no circumstances should the unmasked input values (the *input* argument to the *mask* method) be included in any **Exception** thrown. Exception details are recorded in the engine logs, making them visible to the engine operator and subject to potential disclosure in support bundles. Similarly, exceptions should not simply be re-thrown as **NonConformantDataException** as the original exception's message may contain the sensitive value.

- Whenever possible, configuration problems should be reported in the *validate* or *setup* method, rather than the *mask* method. Waiting until the *mask* method has run to report an error allows the masking job to run, potentially leaving the database table or file partially masked.

- An algorithm should **never** fail in such a way that sensitive values pass through without being masked. In such cases, non-conformant can be reported as described above.

Logging

The extensibility framework provides the capability for an algorithm to create a logger in order to write diagnostic messages to the Delphix Masking Engine logs. **Under no circumstances should unmasked data (any *input* argument values to the *mask* method) be logged**. Logged messages are visible to users via the UI and web API, and may be disclosed in support bundles. It is recommended that production algorithms never log in the *mask* method, for both performance and security reasons.

Additionally, plugin code should *never* read or write any of the *System* input or output streams. Specifically, these are *System.in*, *System.out*, and *System.err*. All logging should be done using the provided logging interfaces.

Handling Secret Credentials and Keys

The JSON document describing the configuration of each algorithm is stored unencrypted on the Delphix Masking Engine and made visible to users with access privileges through the UI and web API. For these reasons, secret values of any kind should **never** be part of an algorithm's configuration, regardless of whether the algorithm is user-created or built into a plugin. This includes secret keys, as well as access credentials or API keys that might be used to access remote systems. The only mechanism available as of release 6.0.3.0 that would allow an algorithm to load a sensitive value without the risk of compromise is reading the value from a file stored on an NFS or CIFS mounted filesystem.

> ✏️ **Note**
>
> A feature to allow plugins to securely access managed credentials will be added in a future Delphix release.

Secret values (keys) or seeds that drive the output "randomization" an algorithm should not be embedded in the algorithm code. Instead, the algorithm's assigned key should be accessed via the CryptoService interface. Static secrets of this kind of risk disclosure should the plugin JAR file be disclosed. There are also risks associated with the algorithm producing the same masking results in all cases, especially if the plugin is to be used for masking by multiple organizations.

Driver Supports

Introduction

As of release 6.0.9.0, the Delphix Masking Engine supports the installation of driver support plugins, written in Java, that provide tasks to execute before/after masking jobs on extended database connectors. Note that this feature requires creating/updating an uploaded JDBC driver to reference the driver support plugin, which is only possible via the web API. Thus creating an extended database connector using that JDBC driver and a corresponding masking job will allow you to enable whatever available tasks that are implemented by the driver support on the job, which you can do via the web API and UI. This process is detailed further here. This feature is referred to as Extensible Driver Supports. This section of the documentation details all aspects of masking driver support plugin usage and development. The *Guided Tour* portion of the workflows section walks the user through the basic process of building a simple plugin and installing it onto the Delphix Masking Engine. Other sections explore topics such as the DriverSupport interface and service interface.

This documentation assumes the reader has some familiarity with Java development as well as operation of the Delphix Masking Engine via both the UI and Web API Client. The reader should also understand the security requirements associated with any new driver supports being developed.

SDK Features

The Extensible SDK provides a number of useful functions that aid development of new driver supports for the Delphix Masking Engine. It is available on the Delphix software download site.

- Creation of empty "skeleton" projects, with build files - the maskScript *init* sub-command
- Testing of the execution of driver support tasks on a database without a masking engine
  - The maskScript *taskExecute* sub-command (**NOTE**: If you want to verify that the **preJobExecute** part of the task was successfully executed, you will want to comment out the reversal of the task in **postJobExecute**, or vice versa. Otherwise, set up your development environment, add a breakpoint and use the debugger to pause after **preJobExecute** execution.)
- Uploading of plugins to the masking engine - the maskScript *install* sub-command
- Sample driver support for MSSQL extended database connector

Getting More Information

Several other sources of information are available to aid in plugin development:

- The README.md file under docs in the Extensible SDK download archive
- The Masking Plugin API Javadoc
- Invoke **maskScript** (located under *sdkTools/bin* in the SDK download) with the -h option for usage help

The DriverSupport Java Interface

Any Java class that should be recognized as a driver support plugin must implement the **DriverSupport** interface. The full details of this interface are described in the Masking Plugin API Javadoc.

Method Overview

This section provides a high-level overview of the methods in the **DriverSupport** interface. For complete details, consult the Masking Plugin API Javadoc included in the Algorithm SDK archive.

- *getTasks* - This method is used to determine the list of available tasks to execute on a corresponding data source. The order in which the tasks are added to the list of tasks indicates the order in which the tasks will be executed on the target data source.

The Life Cycles of Driver Support Objects

The Extensibility framework uses objects classes implementing **DriverSupport** interface for several distinct purposes. These object life cycles are as follows:

Plugin Discovery

This occurs when the extensibility framework evaluates the capabilities present in a **DriverSupport** class.

1. Java object creation - an object of the driver support class is created
2. *getTasks* - determines all available tasks
   - *getTaskName* - get the name of each task
3. Disposal - the Java object is discarded

Driver Support Use

This is the life cycle of a driver object when executing a masking job.

1. Java object creation - an object of the driver support class is created
2. Configuration injection - the masking inventory is used to instantiate a JobInfo object and the database connection is used to instantiate the Connection object (the target SQL connection)
3. *setup* - the *setup* method is called once
4. *preJobExecute* - the *preJobExecute* method is called once before executing the transformation
5. *postJobExecute* - the *postJobExecute* method is called once after executing the transformation
6. Disposal - the Java object is discarded

SDK Workflows

Introduction

This section is intended to walk a developer through several workflows using the Delphix Extensible SDK, such as creating a new algorithm or driver support plugin and installing it on a Delphix Masking Engine.

In order to develop and deploy driver support plugins, you will interact primarily with two tools - the Masking API client, and the Masking Extensible SDK. The Masking API client is a long-standing feature that allows interactive execution of API operations on the Delphix Masking Engine, while the Masking Extensible SDK is a software package created specifically to aid in driver support development.

Outline for a Guided Tour

By following the steps in the outline below, you can tour the basic functionality provided by the Extensible Driver Support feature and Extensible SDK.

1. Create a driver support plugin by choosing one of two options:

   a. Building the sample driver support project

   b. Creating and building your own driver support project

2. Test the driver support plugin using maskScript

3. Install the newly created plugin on the Delphix Masking Engine

4. View and manage the plugins on a Delphix Masking Engine using the API Client

Building the Sample Plugin

The Extensible SDK contains a buildable Sample Driver Support Plugin with a functional driver support illustrating the features of the Extensibility Framework. These simple commands build the plugin containing the sample driver support.

Starting from *sdk_root*:

```
$ cd samples
$ ./gradlew :driverSupport:jar
```

This creates the Sample Driver Support plugin JAR file *sdk_root*/samples/build/libs/driverSupport.jar.

The Sample Driver Support project provides a convenient way to see a working example plugin.

> 🔥 **Tip**
>
> While it is possible to modify these driver supports by changing the Java source and rebuilding the plugin, when starting a new project to develop one, it is highly recommended that you create your own project rather than modifying files in the Sample Driver Support project subtree. This will prevent the loss of customizations to the project build files should you chose to install a new version of Masking Extensible SDK over your existing SDK directory.

Creating a New Project

This section describes how to create a brand new Java project for a new masking driver support plugin. We will use the maskScript utility to create a skeleton project and an empty driver support class in that project.

Creating the Project

Before you begin, you'll want to pick a name for your project, and an **empty** directory (outside of the Masking SDK source tree) where your project will be created. Once you've done this, run this **maskScript** command:

```
$ maskScript init -t driverSupport -d <project path> -n <project name> -a <author name> -v <version>
```

For example, this command will create a project named *demoProject* in the *demo-proj* subdirectory of your home directory.

```
$ maskScript init -t driverSupport -d $HOME/demo-proj -n demoProject -a <plugin author's name> -v <version>
```

For the rest of this section, we'll assume a new project has been created under *proj_dir*. Change your working directory to *proj_dir*. You'll notice that the project is created with a sample driver support file *proj_dir*/src/main/java/com/sample/masking/driverSupport/MSSQLDriverSupport.java. It's possible to build this into a usable plugin by running:

```
$ cd <proj_dir>
$ ./gradlew jar
```

> ⚠️ **Warning**
>
> This sample driver support project is not intended to be used in a production environment and is only meant to serve as an example.

Creating a Driver Support Class

Run the maskScript utility to create a skeleton class file:

```
$ cd <proj_dir>
$ maskScript generate driverSupport -p com.delphix.demo -c <class_name> -s .
```

By convention, the class file .java will be created under a sub-directory path based on the package name, so it might be he helpful to use the find command to locate it:

```
$ find . -name <class_name>.java
./src/main/java/com/delphix/demo/<class_name>.java
```

The initial content of this file is:

```java
package com.delphix.demo;

import com.delphix.masking.api.driverSupport.DriverSupport;
import com.delphix.masking.api.driverSupport.Task;
import com.delphix.masking.api.driverSupport.jobInfo.JobInfo;
import com.delphix.masking.api.provider.ComponentService;
import com.delphix.masking.api.provider.LogService;
import java.sql.Connection;
import java.util.ArrayList;
import java.util.List;

public class <class_name> implements DriverSupport {

        /**
         * This method serves as a directory of Task objects provided by this plugin.
         *
         * @return an ordered list of tasks. The order that tasks are added to the returning list is the
         *      order that they will be executed in.
         */
        @Override
        public List<Task> getTasks() {
                // TODO: return list of implemented task objects
                List tasks = new ArrayList<>();
                tasks.add(new ExampleTask());

                return tasks;
        }

        public class ExampleTask implements Task {
                private JobInfo jobInfo;
                private LogService logService;
                private Connection targetConnection;

                @Override
                public String getTaskName() {
                        return "Example Task";
                }

                @Override
                public void setup(ComponentService serviceProvider) {
                        this.jobInfo = serviceProvider.getJobInfo();
                        this.targetConnection = serviceProvider.getTargetConnection();
                        this.logService = serviceProvider.getLogService();
                }

                @Override
                public void preJobExecute() {
                        // TODO: implement code to execute BEFORE masking job runs.
                }

                @Override
                public void postJobExecute() {
                        // TODO: implement code to execute AFTER masking job runs.
                }
        }
}
```

Implementing the Driver Support Class

The first thing to notice about the skeleton driver support class is that the `getTasks` method just returns an array of tasks with a single no-op task called `ExampleTask` . This means no actual additional transaction will be performed on the target data as part of a masking job, so this will certainly need to change.

It is recommended that you change the task class to a name that more accurately reflects what the task does as well as the string returned from the method `getTaskName` . Delete the `TODO` comments in .java once development is complete.

In order to rebuild the project to generate the driver support plugin JAR, you'll need to first update `settings.gradle` to include the project directory:

```
/*
 * Copyright (c) 2019, 2021 by Delphix. All rights reserved.
 */

pluginManagement {
        resolutionStrategy {
                eachPlugin {
                        if ( requested.id.id == 'com.diffplug.gradle.spotless' ) {
                                useModule( "com.diffplug.spotless:spotless-plugin-gradle:$spotlessVer" )
                        }
                }
        }
}

rootProject.name = '<proj_dir>'
include 'sdkTools'
include 'algorithm'
include 'assemble'
include 'driverSupport'
```

Then to generate the driver support plugin JAR:

```
$ ./gradlew jar
```

This creates or updates the plugin JAR file _proj_dir_/build/libs/.jar

Service Discovery

Java service discovery is used to determine which classes in the plugin JAR present relevant functionality to the Delphix Masking Engine. When a plugin is loaded, the file *com.delphix.masking.api.plugin.DriverSupport* under *META-INF/services* in the JAR is consulted for a list of classes that implement the **DriverSupport** interface.

> ✏ **Note**
>
> When the maskScript *generate* sub-command is used to create a new driver support class, the service discovery metadata file is automatically updated.

If a driver support class is missing from the services file, it will not be usable when the plugin is loaded. It is essentially invisible to the extensibility framework. If a class is mentioned in this file but not present in the JAR, the plugin will fail to load.

Executing a Driver Support Task Using the SDK

It will often be more convenient to use the SDK utilities to test a driver support since this avoids the need to install or update your plugin, create or update a jdbc driver to reference the driver support plugin, and execute jobs on the Delphix Masking Engine. This can be done from the command line using maskScript.

Using maskScript to Test a Driver Support Task

The **maskScript** utility is non-interactive, which lets you execute a task on a given data source. The jdbc driver, driver support and task are selected using command-line options. This example uses the Sample Driver Support plugin. This plugin can be built using the process described here.

Create a task set up json file that corresponds to the specific table and desired database with the contents:

```
{
  "tableMetadata": [
    {
      "name": "Person",
      "schema": "dbo",
      "columns": [
        {
          "name": "column_pk"
        },
        {
          "name": "column_name_1"
        },
        {
          "name": "column_name_2"
        },
        {
          "name": "column_name_3"
        }
      ]
    }
  ],
  "jdbcConnection": {
    "username": "USERNAME",
    "password": "PASSWORD",
    "host": "jdbc:sqlserver://HOST:1433;databaseName=DB_NAME",
    "propertyFilePath": ""
  }
}
```

Execute the task by indicating the name of the desired task, driver support filepath, task set up json, and jdbc driver:

```
$ maskScript taskExecute -n "Task Name" -j /path/to/driverSupport.jar -c /path/to/task-setup.json -l
/path/to/jdbcDriver.jar
```

> ✏️ **Note**
>
> In order to be usable, the class that implements **DriverSupport** must also be listed in the appropriate service description file. Refer to this section for details.

Use any available database management tool like DbVisualizer to connect to the database and verify that the task was successfully executed.

Retrieving Information about Installed Plugins

The GET endpoints are useful for getting information about plugins. After following the steps in this section to install the Sample Driver Support plugin, the GET operation will return (elided for brevity):

```
{
    "pluginId": 9,
    "pluginName": "Sample Plugin",
    "pluginAuthor": "Sample Plugin Author",
    "pluginType": "DRIVER_SUPPORT",
    "originalFileName": "driverSupport.jar",
    "originalFileChecksum": "f8398c0768ecf7709c6992b3f048f9da8be640285b3ccc968973949ca3cceb02",
    "installDate": "2021-04-21T15:29:01.982+00:00",
    "installUser": 5,
    "builtIn": false,
    "pluginVersion": "1.5.0",
    "pluginObjects": [
        {
            "objectIdentifier": "1",
            "objectName": "Disable Constraints",
            "objectType": "DRIVER_SUPPORT_TASK"
        },
        {
            "objectIdentifier": "2",
            "objectName": "Disable Triggers",
            "objectType": "DRIVER_SUPPORT_TASK"
        },
        {
            "objectIdentifier": "3",
            "objectName": "Drop Indexes",
            "objectType": "DRIVER_SUPPORT_TASK"
        }
    ]
},
...
```

> **ℹ Info**
>
> The `objectIdentifier` field refers to the ID of the task. The order in which the tasks are returned from the API is the order in which the tasks will be executed; the `objectIdentifier` (task ID) has no bearing on the task execution order.

For each plugin, the plugin metadata, including `pluginId`, `pluginName` and `originalFileChecksum` are displayed first. This is followed by a list of tasks included in the plugin.

Service Interfaces

Introduction

The Extensible Driver Supports framework makes certain services available to the driver support implementation. This prevents the driver support from having to re-implement code to perform certain routine tasks and facilitates seamless integration with the Masking Engine. This functionality is exposed to the driver support class via the **ComponentService** interface.

Whenever a new Masking driver support instance is required for masking, the extensibility framework first injects any saved configuration, then invokes the objects *setup* method. This method is passed a reference to an object that implements **ComponentService**. The driver support's *setup* method can then use this object to access a number of provider methods:

- *getInstanceName* - Get the name of this instance. Because the instance name it is not typically a configurable field in the driver support, the *getName* method will not correctly return the name of an driver support instance, even after JSON configuration injection. This method will always return the correct instance name as known to the Masking Engine.

- *getTargetConnection* - Gets a java.sql.Connection that is made using the target database connector.

- *getJobInfo* - Gets a jobInfo object, which maps the names of tables, schemas, and columns that are in the masking ruleset.

- *getLogService* - Get a logger object, as described in this section

> 🔥 **Getting More Information**

> Refer to the *com.delphix.masking.api.provider* package in the Javadoc for detailed information.

Accessing Masking Engine Rulesets

The `JobInfo` object represents the database connector's inventory on the masking engine. It contains all of the columns that are going to be masked along with the table and/or schema that they belong to.

Example Driver Support Task

```java
public class DropIndexes implements Task {
        private JobInfo jobInfo;
        private LogService logService;
        private Connection targetConnection;

        @Override
        public String getTaskName() {
                return "Drop Indexes";
        }

        @Override
        public void setup(ComponentService serviceProvider) {
                this.jobInfo = serviceProvider.getJobInfo();
                this.targetConnection = serviceProvider.getTargetConnection();
                this.logService = serviceProvider.getLogService();
        }

        /**
         * This method is to structure all of the columns belonging to the jobInfo.
         *
         * @return A String of comma separated column names.
         */
        private String getCommaSeparatedColumnNames() {
                StringBuilder resultStringBuffer = new StringBuilder();
                for (TableInfo table : jobInfo.getTables()) {
                        resultStringBuffer.append(
                                        table.getColumns().stream()
                                                        .map(ColumnInfo::getName)
                                                        .map(this::singleQuoted)
                                                        .collect(Collectors.joining(",")));
                        resultStringBuffer.append(",");
                }
                String commaSeparatedResult = resultStringBuffer.toString();
                return commaSeparatedResult.substring(0, commaSeparatedResult.length() - 1);
        }
}
```

Some methods have been omitted for brevity.

> ✏️ **Note**
>
> See the Javadocs for further information on the JobInfo, SchemaInfo, TableInfo and ColumnInfo interfaces.

Accessing Database Servers (JDBC)

Driver support plugins will require access to the target database table on which its selected tasks will be run as part of a masking job. The extensible driver support framework allows driver supports to access database servers using JDBC connections, utilizing the existing masking web API. The same connection that is built during the test connection endpoint (`POST /database-connectors/{connector_id}/test`) on the masking engine is the same connection that will be returned by the service provider's **getTargetConnection** method.

Example Driver Support Task

```java
public class DisableTriggers implements Task {
    ...
    private Connection targetConnection;
    ...

    @Override
    public String getTaskName() {
        return "Disable Triggers";
    }

    @Override
    public void setup(ComponentService serviceProvider) {
        this.jobInfo = serviceProvider.getJobInfo();
        this.targetConnection = serviceProvider.getTargetConnection();
        this.logService = serviceProvider.getLogService();
    }

    ...

    @Override
    public void preJobExecute() throws MaskingException {
        long start = System.currentTimeMillis();
        this.triggersOnMaskedTables = findEnabledTriggersOnMaskedTables();
        try (Statement statement = targetConnection.createStatement()) {
            for (Map.Entry<String, String> entry : triggersOnMaskedTables.entrySet()) {
                String triggerName = entry.getKey();
                String tableName = entry.getValue();
                String disableTriggersStatement =
                        String.format(MODIFY_TRIGGERS_SQL, "DISABLE", triggerName, tableName);
                try {
                    statement.execute(disableTriggersStatement);
                } catch (SQLException e) {
                    String errorMessage = ...;
                    logService.error(errorMessage + e);
                    throw new MaskingException(errorMessage, e);
                }
            }
        } catch (SQLException e) {
            String errorMessage = "Error creating a statement on target connection.";
            logService.error(errorMessage + e);
            throw new MaskingException(errorMessage, e);
        }
    }
}
```

Some methods have been omitted for brevity.

Logging

It is possible for a driver support plugin to write information into the app logs, and consequently, Delphix Masking Engine logs. This is accomplished by using calling the *getLogService* method of the **ServiceProvider** interface provided at the driver support setup. The resulting **LogService** object may be used to make logging entries at various levels of severity. The available log levels are ERROR, WARNING, INFO, and DEBUG.

The log interface is provided to allow for debugging output during driver support development, and for reporting of statistical or similar values detailing the overall operation of the driver support, typically in the *tearDown* method.

> 🔥 **Logging Verbosity**
>
> Driver support tasks also should not log progress messages or other verbose details, especially from the **preJobExecute** or **postJobExecute** methods, as this will fill the log files with messages and may impact job performance. There is a rate-limiting mechanism that limits the volume of messages each driver support can write over time, but any amount of routine logging is likely to diminish the overall usefulness of the logs by obscuring more important messages.

Example Code

This example is take from the MSSQL sample Disable Constraints driver support task provided with the SDK:

```java
public class DisableConstraints implements Task {
    ...
    private LogService logService;
    ...

    @Override
    public String getTaskName() {
        return "Disable Constraints";
    }

    @Override
    public void setup(ComponentService serviceProvider) {
        this.jobInfo = serviceProvider.getJobInfo();
        this.targetConnection = serviceProvider.getTargetConnection();
        this.logService = serviceProvider.getLogService();
    }

    ...

    @Override
    public void preJobExecute() throws MaskingException {
        long start = System.currentTimeMillis();
        disableConstraints();
        logService.info(
                String.format(
                        "Total execution to disable all constraints on masked tables took %s ms.",
                        String.valueOf(System.currentTimeMillis() - start)));
    }

    /** This function enables all constraints on the target database table. */
    private void disableConstraints() throws MaskingException {
        this.enabledConstraints = findEnabledConstraints();
        try (Statement statement = targetConnection.createStatement()) {
            for (ConstraintMetadata constraint : enabledConstraints.values()) {
                logService.info(
                        String.format(
                                "Starting to disable constraint: \"%s\" on table \"%s\"",
                                constraint.getName(), constraint.getQualifiedTableName()));
                try {
                    String builtSqlStatement =
                            String.format(
                                    ALTER_CONSTRAINT_STATEMENT,
                                    constraint.getQualifiedTableName(),
                                    constraint.getDisableAction(),
                                    constraint.getName(),
                                    ";");
                    logService.info(builtSqlStatement);
                    statement.execute(builtSqlStatement);
                } catch (SQLException e) {
                    String errorMessage =
                            String.format(
                                    "Error disabling constraint: \"%s\" on table \"%s\".",
                                    constraint.getName(), constraint.getQualifiedTableName());
                    logService.error(errorMessage + e);
                    throw new MaskingException(errorMessage, e);
                }
                logService.info(
                        String.format(
                                "Finished disabling constraint: \"%s\" on table \"%s\".",
                                constraint.getName(), constraint.getQualifiedTableName()));
```

```
            }
        } catch (SQLException e) {
            String errorMessage =
                    String.format(
                            "Error creating statement on target connection %s: ",
                            targetConnection.getClass());
            logService.error(errorMessage + e);
            throw new MaskingException(errorMessage, e);
        }
    }
    ...

    @Override
    public void postJobExecute() throws MaskingException {
        long start = System.currentTimeMillis();
        enableConstraints(); // comment this out if testing of the task execution via the SDK is desired
        logService.info(
                String.format(
                        "Total execution to enable all constraints on masked tables took %s ms.",
                        System.currentTimeMillis() - start));
    }
```

Many methods and fields elided for the sake of brevity

The relevant details here:

- The *setup* method uses the provided **ComponentService** object to get a **LogService** instance, saving it as *logService*.

- The *disableConstraints* method calls the logger's *info* method to write informational messages at random during execution. This kind of "progress" logging may be useful during development but should be removed for driver supports before production deployment. It also calls the logger's *error* method in the event of a failure to connect to the data source or otherwise execute the task on the given data source.

Managing Plugins Using the API Client

The Delphix Masking Engine's web API includes a *plugin* endpoint for managing plugins:

plugin		Show/Hide	List Operations	Expand Operations
GET	/plugin			Get all plugins
POST	/plugin			Install plugin
DELETE	/plugin/{pluginId}			Delete plugin
GET	/plugin/{pluginId}			Get plugin detail by pluginId
PUT	/plugin/{pluginId}			Update plugin

Displaying Information about Installed Plugins

The GET endpoints are useful for getting information about plugins. After following the steps in this section to install the plugin, the GET operation will allow you to retrieve information about the installed plugins. To know what response and information to expect, please see the respective documentation for driver supports and algorithms.

Other Plugin Endpoint Operations

In addition, to GET, the *plugin* endpoint supports the other CRUD operations:

- POST - install a new plugin

- PUT - update an existing plugin

- DELETE - remove a plugin from the system

The POST and PUT operations both require a *fileReference* value representing the plugin file to be installed or updated. These values are the result of using the *fileUpload* endpoint to upload the plugin JAR file to the Masking Engine.

In order to install a new version of this plugin, one could use the PUT operation, or, assuming the algorithm or driver support plugin are not in use, simply DELETE the plugin and POST a new version (or install using the SDK maskScript). Both PUT and DELETE operations require the pluginId value listed for each plugin using the GET operation. Refer to this section for details to help the plugin author ensure that new versions of a plugin can successfully install over an existing version using the PUT operation.

Installing a Plugin onto the Delphix Masking Engine

Once you've successfully built a plugin, it's possible to upload it using the *fileUpload* endpoint in the Masking Engine's API Client, then install the plugin using the *plugin* endpoint. The SDK's **maskScript** includes a sub-command to automate this process. Replace "admin" with your username if you prefer to install the plugin as another user.

```
$ maskScript install -j <path to plugin JAR> -H <engine hostname> -u admin
```

For example, if you've chosen to build the included Sample Algorithm Plugin in its standard location, and the IP address of your Delphix Masking Engine is 10.0.0.1, this command would install the Sample Algorithm Plugin onto your engine:

```
$ maskScript install -j algorithm/build/libs/algorithm.jar -H 10.0.0.1 -u admin
```

You will be prompted for the Delphix Masking Engine user's password.

Upon success, this command will display the JSON response from the API request, including details about the installed plugin as well as a list of the frameworks and algorithms that were installed.

When installing a plugin using the **maskScript**, the -n option may be used to override the plugin name on the Masking Engine. This may be used to install two plugins with the same built-in name on the engine at once (for example, two different versions of the same plugin), but should usually be avoided due to the potential confusion that can result from installing the same plugin on multiple engines with different names.

> ✏️ **Note**
>
> The Web API Client may also be used to manage the plugins installed on the Delphix Masking Engine, as described in this section. Also, algorithms support installing multiple plugins on a masking engine.

Secure Plugin Deployment

It is absolutely vital that only known plugin modules from trusted vendors be installed on the Delphix Masking Engine. A bad plugin may include algorithms that malfunction, possibly by failing to mask data or entering a loop consuming CPU or memory resource. This can lead to job failure, the engine UI becoming unresponsive, or failure to properly mask sensitive data in the case of algorithms. Plugin execution is sandboxed using the Java Security Manager to guard against malfunctioning code. However, JVM security has historically proven susceptible to allowing untrusted modules to run with the danger of malicious code gaining enhanced or full access to the system running the JVM.

With these considerations in mind, this section describes steps the Delphix Masking Engine administrator can take to ensure that only trusted plugins are executed.

Using Roles to Restrict Plugin Installation

This section describes how to define roles and assign roles to Delphix Masking Engine users. The new profile privilege **Plugins** controls which users are able to install new plugins on to the engine. It is advised that only users that **need** the ability to install plugin modules onto the engine be granted roles that include this privilege.

Verifying the SHA256 Hash of Installed Plugins

When the Masking Web API Client *plugin* endpoint is used to GET the details of a plugin, the field *originalFileChecksum* contains the SHA256 hash of the plugin file installed. This may be compared to a vendor-supplied list of known plugin hashes to verify that a plugin installed on the Delphix Masking Engine has not been tampered with.

For example:

```
    {
      "pluginId": 9,
      "pluginName": "demoPlugin",
      "originalFileName": "demoProject.jar",
      "originalFileChecksum": "65053d20874ec7929d219b24bdf98ac5b6f7b06ac6bab59712cf78971be135c9",
      "installDate": "2020-06-24T18:19:42.534+0000",
      "installUser": 5,
      "builtIn": false,
      "pluginVersion": "1.0.0",
      "pluginObjects": [
        {
          "objectIdentifier": "demoPlugin:Clobber",
          "objectName": "demoPlugin:Clobber",
          "objectType": "ALGORITHM"
        },
        {
          "objectIdentifier": "demoPlugin:SampleAlgorithm",
          "objectName": "demoPlugin:SampleAlgorithm",
          "objectType": "ALGORITHM"
        }
      ]
    }
```

Most UNIX like operating systems provide a way to compute the same hash of a file on the command line.

Apple OSX Example:

```
$ shasum -a 256 demoProject.jar
65053d20874ec7929d219b24bdf98ac5b6f7b06ac6bab59712cf78971be135c9  demoProject.jar
```

Ubuntu Linux Example:

```
$ sha256sum demoProject.jar
65053d20874ec7929d219b24bdf98ac5b6f7b06ac6bab59712cf78971be135c9  demoProject.jar
```

At the time this document was written, there are no known means that would allow an attacker to produce a plugin module with different content, but the same SHA256 hash value of a particular file.

Terminology

Algorithm Instance - An algorithm instance is a fully-formed algorithm, which may be assigned to mask data in your masking Inventory. Algorithm instances are uniquely identified by their algorithmName in the Masking API, which is sometimes referred to as "algorithm code" or algorithmCd.

Algorithm Component - This term refers to a Java class within an algorithm plugin that implements the MaskingAlgorithm Java interface.

Algorithm Framework - This term refers to a family of algorithms on the Delphix Masking Engine. It is necessary to create an instance of an algorithm framework in order to use it - for example, FirstNameLookup is an instance of the Secure Lookup (aka. SL) algorithm framework.

Delphix Algorithm SDK - A toolkit authored by Delphix to support the development of algorithm plugins. This includes a CLI for testing algorithms, a skeleton generator for creating empty plugin projects and algorithm classes, and sample algorithms illustrating various use cases.

Delphix Masking API - This refers to the set of web APIs offered by the Delphix Masking Engine over HTTP/HTTPS. This API is sometimes referred to as the V5 APIs (referencing their current major version number) or Masking Web API.

Delphix Masking Plugin API - A package containing the set of Java interfaces that may be implemented in and consumed by a plugin for the Delphix Masking Engine. In order for a plugin to supply algorithms, one or more classes in the plugin must implement the MaskingAlgorithm interfaces provided by this API. This component also includes some common utilities used to load and run plugins on the engine and in the Masking SDK. The JAR containing the appropriate version of the Delphix Masking Plugin API classes has been embedded in the Algorithm SDK zip file.

Plugin - A JAR file containing classes that implement interfaces usable to extend the Delphix Masking Engine. Currently, only masking algorithms may be included in plugins. Plugins also contain self-descriptive metadata to facilitate their use on the engine.

Multi-Column (MC) Algorithm - An algorithm that can take as input more than one field and mask one or all the inputted fields, computing the masked value using any of the fields provided. An MC Algorithm can also take in read-only fields that it does not modify but uses to compute a masked value for another field. The type of the input specified for an MC Algorithm is GENERIC_DATA_ROW, though all the fields must specify one of the "standard" masking types (STRING, BIG DECIMAL, etc).