DELPHIX MASKING · REFERENCIA TÉCNICA

Guía de Algoritmos de Enmascaramiento

Cómo transforma el dato cada algoritmo, con qué parámetros, y qué esperar en la salida.

31	9	62
ALGORITMOS	CATEGORÍAS	INSTANCIAS INTEGRADAS

Plugin `delphix-algorithm-plugin 2026.3.0`

Salidas verificadas en AlgorithmRunner

Índice

Los algoritmos están agrupados según las mismas categorías de la barra lateral del Algorithm Tester. Todos los pares entrada → salida de esta guía se generaron ejecutando el algoritmo de verdad — ninguno es ilustrativo.

PARTE 1

Datos Personales

Algoritmos que reconocen la semántica del campo — correo, teléfono, nombre, CPF/CNPJ — y generan sustitutos sintéticos que siguen pareciendo (y validando como) datos reales.

1.1 Email

`algorithm.plugin.email.Email` **DETERMINISTA**

Enmascara una dirección de correo tratando cada mitad por separado: la parte anterior al `@` (el nombre) y el dominio posterior. Cada mitad tiene su propia estrategia, lo que permite anonimizar por completo al usuario y conservar el dominio — manteniendo la distribución por empresa de la que dependen los informes.

FORMATO DE ENTRADA

> Dirección de correo válida, con `@` y dominio. Ej.: `juan.perez@empresa.com`. Las entradas mal formadas pasan al `errorHandlingAction`.

ENTRADA → SALIDA

ENTRADA		SALIDA
`juan.perez@empresa.com`	→	**`SQRJSH7TWFSNRGVTXM7NTXYUSC4SZROSI2WUT4RCTFHHWJTINV3A@example.com`**
`juan.perez@empresa.com`	→	**`SQRJSH7TWFSNRGVTXM7NTXYUSC4SZROSI2WUT4RCTFHHWJTINV3A@empresa.com`**

La primera fila usa `domainAction: REPLACEMENT`; la segunda, `domainAction: PRESERVE`. Nótese que el nombre generado es idéntico en ambas — depende solo de la entrada y de la clave.

PARÁMETROS

`nameAction` *	Cómo enmascarar la parte del nombre. **UNIQUE** — genera un identificador largo y único por entrada (garantiza que dos correos distintos nunca colisionen). **LOOKUP** — sortea un nombre de un archivo de lookup. **APPLY_ALGORITHM** — pasa el nombre por otro algoritmo. **APPLY_FIRSTNAME_AND_LASTNAME_ALGORITHMS** — divide el nombre y aplica algoritmos de nombre y apellido por separado, produciendo algo como `ana.silva@…`.
`domainAction` *	Cómo tratar el dominio. **REPLACEMENT** — lo cambia por `domainReplacementString`. **APPLY_ALGORITHM** — pasa el dominio por otro algoritmo. **PRESERVE** — conserva el dominio original intacto.
`domainReplacementString`	El dominio sustituto, usado solo cuando `domainAction = REPLACEMENT`. Ej.: `example.com`.
`errorHandlingAction`	Qué hacer cuando la entrada no es un correo válido. **EXCEPTION** — interrumpe con error. **CHARACTER_MAPPING** — aplica un enmascaramiento genérico carácter a carácter. **PRESERVE** — devuelve el valor original sin alterar.

> **Cuándo usar cada uno.** Si el objetivo es solo impedir el envío accidental de correos, `domainAction: REPLACEMENT` hacia un dominio que controles es lo más seguro. Si los analistas necesitan seguir segmentando por empresa, usa `PRESERVE` — pero recuerda que el dominio puede ser identificador en bases pequeñas.

1.2 Phone

`algorithm.plugin.phone.Phone` **DETERMINISTA**

Genera un número de teléfono sintético conservando exactamente el formato original: paréntesis, espacios, guiones y cantidad de dígitos se mantienen en su sitio. Solo cambian los dígitos — y no todos: el algoritmo tiende a conservar los prefijos que identifican el formato (prefijo internacional, el `6` inicial de móvil).

FORMATO DE ENTRADA

> Teléfono en cualquier formato. La puntuación se mantiene posicionalmente en la salida. Ej.: `+34 612 345 678`, `612345678`.

ENTRADA → SALIDA

ENTRADA		SALIDA
+34 612 345 678	→	**+34 619 735 846**
612345678	→	**619735846**

PARÁMETROS

isPhoneUnique *	Cuando es verdadero, el algoritmo garantiza que dos teléfonos de origen distintos nunca produzcan el mismo número enmascarado. Importa cuando el teléfono se usa como clave de deduplicación o de unión. Cuando es falso, las colisiones son posibles pero los números generados resultan más naturales.

1.3 Name

`algorithm.plugin.name.Name` **DETERMINISTA** **REQUIERE ARCHIVO**

Sustituye un nombre simple — solo el nombre de pila, o solo el apellido — por otro extraído de un archivo de lookup. El sorteo es un hash de (valor + clave), así que el mismo nombre siempre da el mismo sustituto: "Juan" será siempre "Mariana" en todas las tablas del proyecto, lo que preserva las uniones por nombre. Para nombres compuestos, usa **Full Name** (§1.4).

FORMATO DE ENTRADA

> Una sola palabra, sin separadores. Ej.: `Juan`, `Pérez`, `María`.

ENTRADA → SALIDA

ENTRADA		SALIDA
Juan	→	**Mariana**
MARÍA	→	**AMANDA**

Con `maskedValueCase: PRESERVE_INPUT` — la entrada en mayúsculas produjo salida en mayúsculas, aunque el archivo contenga "Amanda".

PARÁMETROS

lookupFile *	Archivo de texto con un nombre por línea. Es la fuente de los sustitutos. Cuantas más líneas, menor la probabilidad de que dos nombres distintos reciban el mismo sustituto. Puede contener cualquier lista — nombres de pila, apellidos, nombres masculinos o femeninos por separado.
maskedValueCase	Capitalización del resultado. **PRESERVE_LOOKUP_FILE** — tal como aparece en el archivo. **PRESERVE_INPUT** — copia el patrón de mayúsculas de la entrada. **ALL_LOWER** / **ALL_UPPER** — fuerza minúsculas o mayúsculas.
particlesToRemoveFile	Archivo con partículas (una por línea) que se eliminan de la entrada *antes* de calcular el hash. Hace que "de la Cruz" y "Cruz" produzcan el mismo sustituto en vez de tratarse como nombres distintos.
particlesToPreserveFile	Archivo con partículas que deben reaparecer en el resultado. El sustituto generado recupera las partículas que tenía la entrada original.
maxLengthOfMaskedName	Longitud máxima en caracteres. Los nombres del archivo que superen el límite quedan excluidos del sorteo. Úsalo cuando la columna destino tenga restricción de tamaño y no quieras truncamiento.
maxNumberNames	Máximo de nombres devueltos en modo multivalor. Para una columna simple, déjalo vacío.
filterAccent	Cuando es verdadero, elimina los acentos antes de calcular el hash — haciendo que "José" y "Jose" produzcan el mismo sustituto. Recomendado en bases brasileñas donde la acentuación es inconsistente.
inputCaseSensitive	Cuando es verdadero, "Juan" y "JUAN" son entradas distintas y pueden recibir sustitutos diferentes. El valor por defecto (falso) ignora las mayúsculas al comparar.

1.4 Full Name

`algorithm.plugin.name.FullName` **DETERMINISTA**

Divide un nombre completo en nombre(s) de pila y apellido, y aplica un algoritmo distinto a cada parte. Es un orquestador: no enmascara nada por sí mismo, solo decide dónde termina el nombre y empieza el apellido, delegando cada trozo al algoritmo que le indiques.

FORMATO DE ENTRADA

> Nombre completo separado por espacios. Ej.: `Juan Carlos Pérez`, `María Gómez Ruiz`.

ENTRADA → SALIDA

ENTRADA		SALIDA
`Juan Carlos Pérez García`	→	**`Malachi Fabiano Kishor`**

PARÁMETROS

`lastNameAtTheEnd`	Dónde está el apellido. **true** — es la última palabra ("Juan Carlos Pérez" → apellido "Pérez"). **false** — es la primera (para bases que guardan "Pérez, Juan").
`ifSingleWordConsiderAsLastName`	Desempate para entradas de una sola palabra. **true** — la trata como apellido. **false** — la trata como nombre de pila. Define cuál de los dos algoritmos se invoca.
`firstNameAlgorithmRef`	Algoritmo aplicado al nombre o nombres de pila. Ej.: `dlpx-core:FirstName`. Si se omite, los nombres de pila quedan *sin enmascarar*.
`lastNameAlgorithmRef`	Algoritmo aplicado al apellido. Ej.: `dlpx-core:LastName`. Si se omite, el apellido queda sin enmascarar.
`lastNameSeparators`	Separadores adicionales además del espacio. Ej.: `["-"]` hace que "María-Pérez" se lea como nombre "María" + apellido "Pérez".
`maxLengthOfMaskedName`	Límite de caracteres del nombre completo enmascarado, para respetar el ancho de la columna destino.
`maxNumberFirstNames`	Cuántos nombres de pila enmascarar. Con `1`, "Juan Pedro Carlos Pérez" solo tiene "Juan" enmascarado — "Pedro Carlos" pasa intacto.

> **Cuidado con los valores por defecto.** Ambos `…AlgorithmRef` son opcionales, y cuando quedan vacíos la parte correspondiente **no se enmascara**. Es fácil configurar solo el apellido y dejar que los nombres de pila se filtren a producción — revisa siempre los dos.

1.5 Financial ID BR (CPF/CNPJ)

`algorithm.plugin.financialId.FinancialIdBr` **DETERMINISTA**

Enmascara CPF y CNPJ brasileños generando números sintéticos que siguen siendo **matemáticamente válidos** — los dígitos verificadores se recalculan. El tipo se detecta automáticamente por la cantidad de dígitos: 11 es CPF, 14 es CNPJ. El formato de entrada (puntos, barra, guion) se conserva en la salida.

FORMATO DE ENTRADA

> CPF o CNPJ, con o sin formato. Ej.: `123.456.789-09`, `12345678909`, `11.222.333/0001-81`, `11222333000181`.

ENTRADA → SALIDA

ENTRADA		SALIDA
`123.456.789-09`	→	**`665.927.067-16`**
`11.222.333/0001-81`	→	**`89.572.306/0001-26`**
`11222333000181`	→	**`89572306000126`**
`12345678000190`	→	`Error: dígitos verificadores inválidos (maskInvalidInput = false)`

Las filas 2 y 3 muestran el mismo CNPJ con y sin puntuación: los dígitos generados son idénticos, solo cambia el formato.

PARÁMETROS

`maskInvalidInput`	Qué hacer cuando el CPF/CNPJ de entrada no pasa la validación de dígitos verificadores. **false** (por defecto) — lanza error y detiene. **true** — lo enmascara igualmente, ignorando la invalidez. Las bases heredadas suelen contener documentos inválidos; si es tu caso, actívalo o configura `fallbackAlgorithm`.
`cmNumericRef`	Algoritmo usado internamente para mezclar los dígitos antes de recalcular el verificador. Si queda vacío, usa el Character Mapping numérico integrado. Indica una instancia del catálogo, ej.: `dlpx-core:CM Digits`.
`fallbackAlgorithm`	Algoritmo invocado cuando el valor no se reconoce ni como CPF ni como CNPJ — campo en blanco, texto libre, longitud inesperada. Sin él, esos casos se convierten en errores.

PARTE 2

Fechas

Cuatro estrategias distintas para fechas: desplazar preservando intervalos, desplazar por valores discretos, sustituir por completo, o simplemente acotar a un rango.

2.1 Date Shift

`algorithm.plugin.dateAlgorithms.DateShift` **DETERMINISTA**

Desplaza la fecha una cantidad sorteada dentro de `[minRange, maxRange]`. El sorteo es determinista por la clave, de modo que la misma fecha siempre recibe el mismo desplazamiento — lo que **preserva el orden cronológico** entre registros. Es el algoritmo indicado cuando los análisis dependen de intervalos ("días entre pedido y entrega").

FORMATO DE ENTRADA

> ISO 8601: `yyyy-MM-dd` o `yyyy-MM-ddTHH:mm:ss`. La salida siempre incluye componente de hora.

ENTRADA → SALIDA

ENTRADA		SALIDA
2024-03-15	→	**2024-12-21T00:00**
2024-03-15T14:30:00	→	**2023-03-21T14:30**
2024-01-31	→	**2023-02-28T00:00**

Las dos primeras con `±365 DAYS`; la tercera con `±12 MONTHS`. Nótese que la hora original siempre se conserva.

PARÁMETROS

minRange	Desplazamiento mínimo, en unidades de `unit`. Negativo permite retroceder. Ej.: `-365` con `DAYS` permite volver un año atrás.
maxRange	Desplazamiento máximo. Positivo permite avanzar. Un rango estrecho preserva mejor la plausibilidad de los datos; uno amplio aumenta la protección.
unit	Unidad del desplazamiento: `SECONDS`, `MINUTES`, `HOURS`, `DAYS`, `MONTHS`, `YEARS`.
roll	Solo relevante con `MONTHS` o `YEARS`, cuando el resultado caería en una fecha inexistente (31 de enero + 1 mes = 31 de febrero). **true** — encoge al último día válido del mes (28 de febrero). **false** — desborda al mes siguiente (3 de marzo). En caso de duda, usa `false`.

> **Sobre la preservación de intervalos.** Como el desplazamiento depende de la fecha de origen, dos registros con fechas distintas reciben desplazamientos distintos — la distancia entre ellos **no** se preserva exactamente. Si mantener las distancias intactas es un requisito, el desplazamiento debe ser fijo por entidad, que es lo que ofrece `dlpx-core:Date Shift Fixed`.

2.2 Date Shift Discrete

`algorithm.plugin.dateAlgorithms.DateShiftDiscrete` **DETERMINISTA**

Variante de Date Shift en la que el desplazamiento no procede de un rango continuo, sino de un **conjunto finito de valores** definido en un archivo de configuración. Útil cuando el negocio exige desplazamientos estandarizados — por ejemplo, solo múltiplos de 7 días, para preservar el día de la semana.

FORMATO DE ENTRADA

> ISO 8601: `yyyy-MM-dd` o `yyyy-MM-ddTHH:mm:ss`.

ENTRADA → SALIDA

ENTRADA		SALIDA
2024-03-15	→	**2024-03-10T00:00**

PARÁMETROS

Sin parámetros expuestos en el esquema — el conjunto de desplazamientos proviene de la configuración integrada del plugin.

2.3 Date Replacement

`algorithm.plugin.dateAlgorithms.DateReplacement` **DETERMINISTA**

Descarta la fecha original y sortea una fecha **absoluta** dentro de `[minDate, maxDate]`. A diferencia de Date Shift, aquí no hay relación alguna entre entrada y salida más allá del determinismo — el orden cronológico original se destruye. Úsalo cuando la fecha en sí es sensible y ningún análisis temporal depende de ella.

FORMATO DE ENTRADA

> ISO 8601 con hora: `yyyy-MM-ddTHH:mm:ss`. Ej.: `1985-07-20T00:00:00`.

ENTRADA → SALIDA

ENTRADA		SALIDA
1985-07-20T00:00:00	→	**1999-07-26T00:00**

PARÁMETROS

minDate	Fecha mínima posible en la salida, con formato `yyyy-MM-ddTHH:mm:ss`. Define el suelo de la ventana de sorteo.
maxDate	Fecha máxima posible, mismo formato. Elige una ventana plausible para el dominio — fechas de nacimiento entre 1970 y 2000, por ejemplo.
unit	Granularidad del sorteo: `SECONDS`, `MINUTES`, `HOURS` o `DAYS`. Con `DAYS`, la componente de hora queda en cero.

2.4 Min/Max Date/Time

`algorithm.plugin.minMax.MinMaxLocalDateTime`

No es exactamente un enmascaramiento, sino un **acotamiento de rango**: las fechas dentro del rango pasan intactas; las de fuera se llevan al límite más cercano. El uso típico es suprimir valores atípicos que identifican individuos — el cliente centenario, la fecha de 1900 usada como centinela.

FORMATO DE ENTRADA

> ISO 8601. Ej.: `2024-03-15T14:30:00`.

ENTRADA → SALIDA

ENTRADA		SALIDA
`1985-06-15T08:00:00`	→	**`1985-06-15T08:00`**
`1950-01-01T00:00:00`	→	**`1970-01-01T00:00`**
`2024-05-10T12:00:00`	→	**`2000-12-31T23:59:59`**

Rango `1970-01-01` a `2000-12-31T23:59:59`. La primera fecha está dentro y pasa sin cambios; la segunda es anterior al suelo; la tercera, posterior al techo.

PARÁMETROS

`minDate` *	Suelo del rango. Cualquier fecha anterior se sustituye por este valor.
`maxDate` *	Techo del rango. Cualquier fecha posterior se sustituye por este valor.
`nonConformingDataDefaultValue`	Valor devuelto cuando la entrada no es una fecha/hora válida. Si queda vacío, el algoritmo lanza un error de dato no conforme y detiene el job.

> **Esto no anonimiza.** Los valores dentro del rango salen **idénticos** a la entrada. Min/Max es una herramienta de supresión de atípicos, no de enmascaramiento — nunca lo uses solo en una columna sensible.

PARTE 3

Numérico

Transformaciones sobre valores numéricos: mapeo determinista a un rango, acotamiento de intervalo, expresiones Java arbitrarias y una regla fija de repetición de dígitos.

3.1 Numeric Mapping

`algorithm.plugin.characterMapping.NumericMapping` **DETERMINISTA**

Mapea un número a otro dentro de `[minValue, maxValue]`, de forma determinista. Cada valor de origen recibe consistentemente el mismo destino, lo que preserva uniones y conteos distintos — pero ni el orden ni la magnitud.

FORMATO DE ENTRADA

> Número entero positivo, sin puntuación ni decimales. Ej.: `12345`.

ENTRADA → SALIDA

ENTRADA		SALIDA
12345	→	**49421**
98765	→	**86050**
11111	→	**67960**

PARÁMETROS

minValue	Menor valor que puede tomar la salida. Elige un suelo que respete la semántica de la columna (ej.: `10000` para garantizar siempre 5 dígitos).
maxValue	Mayor valor de la salida. La amplitud del rango determina la probabilidad de colisión: los rangos estrechos hacen que valores distintos colisionen en el mismo destino.

3.2 Min/Max BigDecimal

`algorithm.plugin.minMax.MinMaxBigDecimal`

La contraparte numérica de Min/Max Date/Time: acota el valor a un rango, dejando pasar intacto lo que ya está dentro. Sirve para aplanar valores atípicos — sueldos muy altos, saldos negativos extremos — que por sí solos identifican a la persona.

FORMATO DE ENTRADA

> Número decimal o entero. Ej.: `5000.50`, `3.14`.

ENTRADA → SALIDA

ENTRADA		SALIDA
5000.50	→	**5000.50**
250	→	**1000**
15000	→	**9999**

Rango `1000` – `9999`. Nótese que la escala decimal del valor dentro del rango se conserva.

PARÁMETROS

`minValue` *	Límite inferior. Los valores menores se elevan a este número.
`maxValue` *	Límite superior. Los valores mayores se bajan a este número.
`nonConformingDataDefaultValue`	Valor devuelto cuando la entrada no es numérica. Vacío hace que el algoritmo lance un error de dato no conforme.

> **Esto no anonimiza.** Igual que el Min/Max de fechas, los valores dentro del rango salen idénticos. Combínalo con otro algoritmo si la columna es sensible.

3.3 Numeric Expression

`algorithm.plugin.expression.NumericExpression`

El algoritmo más abierto del conjunto: evalúa una **expresión Java de una línea** sobre el valor de entrada. La expresión recibe dos variables — `input` (el valor como `BigDecimal`) y `seed` (un `long` derivado de la clave, para enmascaramiento determinista) — además de las constantes que declares.

FORMATO DE ENTRADA

> Número entero o decimal. Ej.: `1000`, `3.14`.

ENTRADA → SALIDA

ENTRADA · EXPRESIÓN		SALIDA
`1234.56` `input.setScale(0, HALF_UP)`	→	**`1235`**
`99.4` `input.setScale(0, HALF_UP)`	→	**`99`**
`1234.56` `new BigDecimal(seed % 9000 + 1000)`	→	**`5556`**
`1000` `input.multiply(new BigDecimal(taxa))`	→	**`800.000000000000044408920985006261616945266723632812500`**

PARÁMETROS

`expression` *	Expresión Java que devuelve `BigDecimal`. Variables disponibles: `input` y `seed`. Las clases necesitan nombre totalmente cualificado — `new java.math.BigDecimal(...)`, `java.math.RoundingMode.HALF_UP`.
`inputType`	Cómo interpretar la entrada antes de exponerla en `input`: `DOUBLE`, `LONG` o `BIG_DECIMAL` (por defecto).
`constants`	Lista de constantes con nombre accesibles en la expresión. Cada una tiene `name` y `value` (siempre string). Ej.: `name="taxa"`, `value="0.8"`, usada como `new java.math.BigDecimal(taxa)`.
`nonConformingDataDefaultValue`	Valor devuelto cuando la entrada no es numérica. Vacío lanza un error.

> **Cuidado con el punto flotante.** La cuarta fila del ejemplo devolvió `800.000000000000444…` en lugar de `800`: la constante `"0.8"` pasó por un `double` antes de convertirse en `BigDecimal`. Para aritmética exacta, usa el constructor de string — `new java.math.BigDecimal("0.8")` — y cierra con `.setScale(2, java.math.RoundingMode.HALF_UP)`.

3.4 Repeat First Digit

`algorithm.plugin.repeatFirstDigit.RepeatFirstDigit`

Regla fija y sin configuración: toma los **4 dígitos finales** del número y los sustituye todos por el primero de ellos, repetido cuatro veces. El resto del número queda intacto. Es un enmascaramiento débil, adecuado para campos de baja sensibilidad donde solo se quiere romper el valor exacto.

FORMATO DE ENTRADA

> Cadena numérica de exactamente **4, 9, 10 o 14 dígitos**, sin puntuación. Otras longitudes se rechazan.

ENTRADA → SALIDA

ENTRADA		SALIDA
1234	→	**1111**
123456789	→	**123456666**

En el segundo caso los 5 primeros dígitos se conservan y los 4 últimos (`6789`) pasan a `6666` .

PARÁMETROS

Ninguno. El comportamiento es fijo.

PARTE 4

String

Los algoritmos de propósito general para texto estructurado. Aquí están las herramientas más usadas del plugin — y las más configurables.

4.1 Character Mapping

`algorithm.plugin.characterMapping.CharacterMapping` **DETERMINISTA**

Cambia cada carácter por otro **del mismo grupo**. Tú defines los grupos (minúsculas, mayúsculas, dígitos...) y el algoritmo garantiza que una letra pase a letra y un dígito a dígito, preservando la longitud y el "formato visual" del dato. Los caracteres que no pertenecen a ningún grupo — espacios, acentos, puntuación — pasan intactos.

FORMATO DE ENTRADA

> Cualquier cadena. Ej.: `Juan Núñez`, `ABC123`.

ENTRADA → SALIDA

ENTRADA		SALIDA
`Juan Núñez123`	→	**`Lqxs Fúñza879`**
`ABC123`	→	**`KCT250`**
`María`	→	**`Qpyík`**
`Juan Núñez123` `(otra clave)`	→	**`Eraf Zúñqf618`**

La `ñ` y el espacio sobrevivieron — no están en ningún grupo configurado. La última fila muestra el efecto de la clave: misma entrada, clave distinta, resultado distinto.

PARÁMETROS

`characterGroups`	Lista de cadenas; cada una es un conjunto de caracteres intercambiables entre sí. Ej.: `["abcdefghijklmnopqrstuvwxyz", "0123456789"]` crea dos grupos aislados — las letras solo pasan a letras, los dígitos solo a dígitos. Los caracteres ausentes de todos los grupos nunca se enmascaran.
`caseSensitive`	Cuando es verdadero, mayúsculas y minúsculas se tratan como grupos separados, preservando la capitalización original. Requiere que declares ambos conjuntos por separado.
`minMaskedPositions`	Número mínimo de posiciones que deben cambiar efectivamente. Impide que el algoritmo devuelva un valor casi idéntico por coincidencia del sorteo.
`preserveRanges`	Lista de tramos a mantener intactos, cada uno con `start`, `length` y `direction`. Úsalo para preservar prefijos o sufijos con significado — código de sucursal, dígito de control.
`preserveLeadingZeros`	Cuando es verdadero, los ceros a la izquierda se mantienen. Esencial en códigos numéricos guardados como texto, donde `007` y `7` no son equivalentes.

> **Por qué los acentos no cambian.** Si los caracteres acentuados deben enmascararse, inclúyelos explícitamente en un grupo: `"aáàâãeéêiíoóôõuú"`. De lo contrario quedan visibles en el dato enmascarado y pueden ayudar a reidentificar el registro.

4.2 Character Replacement

`algorithm.plugin.characterReplacement.CharacterReplacement`

Aplica reglas de sustitución carácter a carácter: cada regla declara *qué* caracteres detectar y *por cuál* cambiarlos. A diferencia de Character Mapping, aquí la sustitución es fija (no sorteada) — todos los caracteres filtrados pasan al mismo símbolo. Puede ejecutarse opcionalmente después de otro algoritmo, mediante `stringAlgorithm`.

FORMATO DE ENTRADA

> Cualquier cadena. Ej.: `Un texto de ejemplo`.

ENTRADA → SALIDA

ENTRADA		SALIDA
`Un texto de ejemplo`	→	`*k b*p** q* *f*ns**`
`María Jiménez`	→	`K*kâ* Y*sæt*f`
`Juan Núñez` `(filterAccents: INPUT)`	→	`B**n **v*y`

Regla: vocales → `*`. Las consonantes también cambiaron porque el algoritmo aplica un enmascaramiento propio antes de las reglas. En la tercera fila, `filterAccents: INPUT` convirtió `ú` y `ñ` a ASCII antes de procesar.

PARÁMETROS

`stringAlgorithm`	Algoritmo aplicado a la entrada *antes* de las reglas de sustitución. Su resultado es lo que pasa por `replacementRules`. Permite componer: enmascarar con un algoritmo fuerte y luego normalizar caracteres problemáticos.
`replacementRules`	Lista de reglas aplicadas en secuencia. Cada regla es un par (conjunto de caracteres a detectar, carácter sustituto).
↳ **`filteredCharacters`**	Los caracteres que esta regla detecta. Forma simple, usada cuando entrada y salida comparten el mismo conjunto.
↳ **`replacementCharacter`**	El carácter único que sustituye cada ocurrencia encontrada.
↳ **`inputFilteredCharacters`**	Forma avanzada: caracteres detectados en la entrada, cuando quieres conjuntos distintos para lectura y escritura.
↳ **`inputReplacementCharacter`**	Sustituto aplicado a las coincidencias de `inputFilteredCharacters`.
↳ **`outputFilteredCharacters`**	Caracteres que, si aparecen en la *salida* de `stringAlgorithm`, también deben sustituirse.
↳ **`outputReplacementCharacter`**	Sustituto aplicado a las coincidencias de `outputFilteredCharacters`.
`requireInputChange`	Cuando es verdadero, exige que al menos una sustitución haya ocurrido — si ninguna regla cambió nada, el algoritmo lanza error. Protege contra configuraciones que silenciosamente no enmascaran nada.
`filterAccents`	Cuándo eliminar acentos. **INPUT** — antes de aplicar las reglas. **OUTPUT** — en el resultado final. **BOTH** — en ambas etapas. **NONE** — nunca (por defecto).

4.3 Segment Mapping

`algorithm.plugin.segmentMapping.SegmentMapping` **DETERMINISTA**

Divide la cadena en segmentos de longitud fija y trata cada uno de forma independiente — enmascarar, preservar o sustituir por una constante. Es el algoritmo indicado para documentos de formato rígido donde las partes tienen significados distintos: CPF, CNPJ, código postal, código de barras, número de cuenta con sucursal incorporada.

FORMATO DE ENTRADA

> Cadena de formato fijo. Con `autoIgnoreCharacters` activo, los separadores (puntos, guiones, barras) se detectan y reposicionan automáticamente. Ej.: `123.456.789-09`.

ENTRADA → SALIDA

ENTRADA		SALIDA
`123.456.789-09`	→	**`633.492.689-81`**
`987.654.321-00`	→	**`576.698.423-45`**

Segmentos de 3-3-3-2 dígitos, todos `MASK_NUMERIC`, con los separadores conservados en sus posiciones originales.

PARÁMETROS

`segments` *	Lista ordenada de segmentos, en la secuencia en que aparecen en el valor (sin contar los caracteres ignorados). La suma de las longitudes debe coincidir con el tamaño del dato.
↳ **`length`** *	Cuántos caracteres ocupa este segmento. Los caracteres ignorados no cuentan.
↳ **`segmentType`** *	**MASK_NUMERIC** — sustituye por dígitos. **MASK_ALPHANUMERIC** — sustituye por caracteres alfanuméricos. **PRESERVE** — mantiene el original. **CONSTANT** — sustituye por el valor fijo de `maskValues`.
↳ **`inputValues`**	Conjunto de caracteres aceptados en este segmento en la entrada, como cadena continua. Ej.: `"0123456789"`. Vacío acepta cualquier carácter.
↳ **`maskValues`**	Repertorio de caracteres usado en la sustitución. Ej.: `"123456789"` (sin el cero) impide que el segmento empiece por `0`. Vacío usa el repertorio por defecto del tipo.
`ignoreCharacters`	Códigos ASCII de los separadores a saltar al contar posiciones — `46` punto, `45` guion, `47` barra. Vuelven a la salida en sus posiciones originales.
`autoIgnoreCharacters`	Cuando es verdadero, detecta por sí solo todos los caracteres no alfanuméricos y los ignora. Evita listar códigos ASCII a mano — recomendado para CPF, CNPJ y códigos postales.
`allowShortSegments`	Cuando es verdadero, acepta entradas más cortas que la suma de los segmentos, procesando lo que haya. Por defecto (falso) lanza error.
`processPreserveBeforeIgnore`	Orden de operación entre eliminar ignorados y calcular los segmentos `PRESERVE`. Actívalo cuando un separador cae dentro de un segmento preservado y las posiciones salen desplazadas.

4.4 Regex Decompose

`algorithm.plugin.decompose.RegexDecompose`

El algoritmo más flexible para texto estructurado. Defines patrones regex; el primero que coincida con la cadena **entera** se aplica, y cada grupo de captura (paréntesis) recibe una acción independiente — preservar, borrar, redactar o pasar por otro algoritmo. El texto *entre* grupos se preserva automáticamente.

FORMATO DE ENTRADA

> Cualquier cadena. Los patrones se prueban en el orden declarado hasta que uno coincida. Si ninguno coincide, se aplica `fallbackAction`.

ENTRADA → SALIDA

ENTRADA · REGEX		SALIDA
`usuario@empresa.com` `([^@]+)(@[^@]+) → REDACT, PRESERVE`	→	`***@empresa.com`
`000001999191111` `(\d{6})(\d{9}) → PRESERVE, REDACT "X"`	→	`000001XXXXXXXXX`
`"000001999191111 "` `mismo patrón, trimInput: true`	→	`"000001XXXXXXXXX "`
`123456789` `(\d{3})(\d+) → PRESERVE, TRUNCATE`	→	`123`
`ABC` `sin fallbackAction`	→	`Error: No pattern matched and no fallbackAction is defined`
`ABC` `fallback REDACT "[INVÁLIDO]"`	→	`[INVÁLIDO]`

PARÁMETROS

`maskPatterns`	Lista de patrones probados **en orden**. Gana el primero cuyo regex coincida con la cadena entera — los demás se ignoran. Esta ordenación es lo que permite el enrutamiento condicional (ver la nota abajo).
↳ **`regex`**	Expresión regular que debe coincidir con la cadena **entera** (anclada implícitamente). Los grupos de captura delimitan las partes que reciben acciones.
↳ **`actions`**	Una acción por grupo de captura, en el mismo orden que los paréntesis. Dos grupos exigen exactamente dos acciones.
↳ **`actions.type`**	**PRESERVE** — mantiene el tramo. **TRUNCATE** — lo elimina (queda cadena vacía). **REDACT** — lo sustituye por texto o carácter fijo. **APPLY_ALGORITHM** — lo pasa por otro algoritmo.
↳ **`actions.redactString`**	Texto fijo que sustituye el grupo *entero*, independientemente del tamaño original. Ej.: `"***"`. Solo con `REDACT`.
↳ **`actions.redactCharacter`**	Un carácter único que sustituye *cada* carácter del grupo, preservando la longitud. Ej.: `"X"` convierte 9 dígitos en `XXXXXXXXX`. Mutuamente excluyente con `redactString`.
↳ **`actions.algorithm`**	Instancia de algoritmo aplicada al grupo cuando el tipo es `APPLY_ALGORITHM`. Acepta nombres del catálogo integrado (apéndice B) o pruebas guardadas en el Tester.
`fallbackAction`	Acción aplicada al valor entero cuando ningún patrón coincide. Tiene los mismos cuatro tipos, con sus propios `redactString`, `redactCharacter` y `algorithm`.

`trimInput`	Elimina los espacios de ambos extremos *antes* de intentar la coincidencia. El relleno se restaura en la salida (ver la tercera fila de la tabla). Esencial para columnas `CHAR(n)`, que llegan rellenadas con espacios a la derecha.
`requireMask`	Cuando está activo, lanza error si ningún patrón coincide, en lugar de usar el fallback. Úsalo cuando los valores fuera de formato deban rechazarse en vez de seguir adelante.
`maxInputLength`	Longitud máxima aceptada en la entrada. Los valores mayores generan error. Vacío no impone límite.

Dos errores frecuentes.
① `NullPointerException … action.type is null` — al añadir una acción desde el formulario, el campo **Tipo de acción** empieza vacío; selecciónalo explícitamente.
② `Fallback action may not be PRESERVE when requireMask is set` — `requireMask` tiene por defecto **true**; para usar `fallbackAction: PRESERVE`, define `requireMask: false` explícitamente.

Enrutamiento condicional por contenido. Como los patrones se prueban en orden, puedes desviar el valor a algoritmos distintos según lo que contenga. Para aplicar un algoritmo cuando hay `@` y otro cuando no:

```
{"maskPatterns": [
  { "regex": "(.+@.+)",
    "actions": [{"type":"APPLY_ALGORITHM","algorithm":{"name":"dlpx-core:Email
SL"}}] },
  { "regex": "(.+)",
    "actions": [{"type":"APPLY_ALGORITHM","algorithm":{"name":"dlpx-
core:FirstName"}}] }
]}
```

El primer patrón solo coincide con valores que tengan `@`; todo lo demás cae en el segundo, que actúa como catch-all.

4.5 String Algorithm Chain

`algorithm.plugin.stringAlgorithmChain.StringAlgorithmChain`

Encadena algoritmos en secuencia: la salida de uno es la entrada del siguiente. Exige un mínimo de dos. Sirve para componer transformaciones que ningún algoritmo aislado ofrece — enmascarar y luego normalizar, o aplicar dos capas de sustitución.

FORMATO DE ENTRADA

> Cadena compatible con el *primer* algoritmo de la cadena. Los siguientes reciben lo que produjo el anterior.

ENTRADA → SALIDA

ENTRADA		SALIDA
`Juan Perez`	→	**Kokou**

Cadena `FirstName` → `LastName` : el primer algoritmo produjo un nombre, y el segundo trató ese resultado como si fuera un apellido.

PARÁMETROS

`algorithmReferences`	Lista ordenada de referencias a instancias, **mínimo 2**. Ej.: `[{"name":"dlpx-core:FirstName"}, {"name":"dlpx-core:LastName"}]` . Los nombres provienen del catálogo integrado (apéndice B) o de pruebas guardadas en el Tester.

> **El orden importa, y el encaje también.** Cada algoritmo debe aceptar el formato que produjo el anterior. Encadenar un algoritmo de fecha después de uno de nombre, por ejemplo, falla — el segundo recibe texto que no puede interpretar.

4.6 Shuffle

`algorithm.plugin.shuffle.Shuffle` ⚠ NO DETERMINISTA MODO LOTE

Redistribuye los valores **entre** los registros: cada fila recibe un valor que pertenecía a otra. No se inventa ningún valor — el conjunto sigue idéntico, solo cambian las asociaciones. Esto preserva perfectamente la distribución estadística de la columna, lo que lo hace ideal para columnas usadas en agregaciones.

FORMATO DE ENTRADA

> Varios valores a la vez (modo lote). Cada valor debe tener al menos `minimumShuffleSize` caracteres.

ENTRADA → SALIDA

LOTE DE ENTRADA		LOTE DE SALIDA
María García	→	**Carlos Romero**
Juan López	→	**Pedro Martínez**
Pedro Martínez	→	**María García**
Ana Sánchez	→	**Juan López**
Carlos Romero	→	**Ana Sánchez**

PARÁMETROS

`minimumShuffleSize`	Longitud mínima para que un valor participe en el barajado (por defecto `3`). Los valores más cortos quedan fuera y pasan sin alterar.

> **No es determinista — a propósito.** La permutación cambia en cada ejecución. Es una decisión de seguridad: una permutación reproducible permitiría reejecutar el barajado y deshacer la anonimización. La consecuencia práctica es que Shuffle **no preserva la integridad referencial** entre tablas ni entre ejecuciones.

PARTE 5

Texto libre

Para campos de texto libre — observaciones, informes, comentarios — donde el dato sensible está incrustado en una prosa que debe seguir siendo legible.

5.1 Free Text Redaction

`algorithm.plugin.freeTextRedaction.FreeTextRedaction`

Recorre un texto libre buscando entidades sensibles — por expresión regular y/o por una lista de términos en un archivo — y sustituye solo los tramos encontrados. Todo lo demás se preserva, manteniendo el documento legible y útil para el análisis.

FORMATO DE ENTRADA

> Texto libre de cualquier tamaño. Ej.: `Contacta con Juan en juan@empresa.com o en el 555-0100.`

ENTRADA → SALIDA

ENTRADA		SALIDA
`Contacta con Juan por correo en juan.perez@empresa.com.`	→	**`Contacta con Juan por correo en [REDACTED].`**

PARÁMETROS

`regularExpressions`	Lista de objetos `{patternString}` con las regex que identifican los tramos a redactar. Cada patrón se aplica a todo el texto, en todas las ocurrencias.
`isDenyList`	**true** — lista de bloqueo: redacta lo que *coincide* con los patrones. **false** — lista de permitidos: redacta todo lo que *no* coincide. La segunda opción es más segura para textos impredecibles, pero suele destruir la legibilidad.
`regExRedactValue`	Texto que sustituye los tramos capturados por las expresiones regulares. Ej.: `[REDACTED]`.
`lookupFile`	Archivo con términos literales, uno por línea, buscados en el texto. Útil para nombres propios y términos que la regex describe mal.
`lookupFileRedactValue`	Texto que sustituye las ocurrencias encontradas por el archivo. Es independiente de `regExRedactValue`, lo que permite marcar visualmente el origen de cada redacción.

5.2 Redact

`algorithm.plugin.redact.Redact`

Versión más simple y directa de la redacción: un mapa de `regex → texto de sustitución`. Cada clave del mapa es un patrón, y el valor correspondiente es lo que ocupa su lugar. Sin ninguna regex configurada, devuelve el valor de entrada sin enmascaramiento alguno.

FORMATO DE ENTRADA

> Cualquier cadena. Los tramos que coincidan con las regex se sustituyen.

ENTRADA → SALIDA

ENTRADA		SALIDA
`Contacta con Juan por correo en juan.perez@empresa.com.`	→	**`Contacta con Juan por correo en [EMAIL].`**

PARÁMETROS

`regexRedact`	Mapa de `{regex → texto_sustitución}`. Permite marcadores distintos por tipo de dato — `[EMAIL]`, `[CPF]`, `[TELÉFONO]` — en una sola configuración.

> **Una configuración vacía no enmascara nada.** Sin ninguna regex en `regexRedact`, el algoritmo devuelve la entrada intacta y **no** emite ninguna advertencia. Confirma siempre que el mapa se rellenó.

PARTE 6

Financiero

Instrumentos financieros con estructura verificable: tarjetas con dígito Luhn e IBAN con checksum de país.

6.1 Payment Card

`algorithm.plugin.characterMapping.PaymentCard` **DETERMINISTA**

Enmascara números de tarjeta preservando el **BIN** (los primeros dígitos, que identifican la marca y el emisor) y, opcionalmente, los últimos dígitos usados en conciliaciones. El número generado mantiene un **dígito verificador Luhn válido**, así que sigue pasando las validaciones de formulario y de sistema.

FORMATO DE ENTRADA

> Número de tarjeta con o sin espacios/guiones. Ej.: `4111 1111 1111 1111`.

ENTRADA → SALIDA

ENTRADA		SALIDA
4111111111111111	→	**4111687664762392**
5500000000000004	→	**5500767862953716**
4111111111111111 `(preserve: 0)`	→	**1834326703509900**

En las dos primeras el BIN (`4111` , `5500`) se preservó automáticamente. En la tercera, con `preserve: 0` , hasta el BIN cambió — la marca de la tarjeta deja de ser identificable.

PARÁMETROS

`preserve`	Cuántos dígitos del *final* de la tarjeta preservar. Habitualmente `4` , para mantener los cuatro últimos que aparecen en extractos y pantallas de confirmación.
`minMaskedPositions`	Número mínimo de posiciones que deben cambiar efectivamente. Garantiza que la tarjeta enmascarada no quede demasiado parecida a la original.

> **Preservar los 4 últimos tiene un costo.** Los últimos cuatro más el BIN más la fecha de transacción suelen bastar para reidentificar una tarjeta en bases pequeñas. Usa `preserve: 4` solo cuando el proceso de negocio realmente dependa de ello.

6.2 IBAN

`algorithm.plugin.iban.IBAN` **DETERMINISTA**

Enmascara un IBAN preservando el código de país y recalculando los dígitos de control, de modo que el resultado sigue siendo un IBAN estructuralmente válido. Tú controlas cuántos caracteres de la parte nacional se mezclan.

FORMATO DE ENTRADA

> IBAN en formato estándar, sin espacios. Ej.: `GB29NWBK60161331926819` .

ENTRADA → SALIDA

ENTRADA		SALIDA
`GB29NWBK60161331926819` `(mask 20)`	→	**`GB61PMNY26253473886483`**
`GB29NWBK60161331926819` `(mask 8)`	→	**`GB74NWBK60161340532625`**

Con `numCharsToMask: 8` el código de banco (`NWBK`) y la sucursal sobreviven; con `20` , prácticamente solo queda el país. El `GB` siempre se preserva y el checksum se recalcula.

PARÁMETROS

`validateInput` *	Cuando es verdadero, valida el checksum del IBAN antes de enmascarar y rechaza las entradas inválidas. Desactívalo solo si la base contiene IBAN sabidamente mal formados que aun así deben procesarse.
`numCharsToMask` *	Cuántos caracteres mezclar, contados desde el final de la parte nacional. Los valores bajos preservan banco y sucursal; los altos enmascaran todo lo posterior al país.

PARTE 7

Lookup y Mapeo

Algoritmos que sustituyen valores consultando una fuente externa — un archivo de sustitutos, una tabla de equivalencias, o una base de mapeos persistente.

7.1 Secure Lookup

`algorithm.plugin.secureLookup.SecureLookup` **DETERMINISTA** **⚠ SALVO CON RANDOMIZE** **REQUIERE ARCHIVO**

Sustituye el valor por una línea de un archivo de lookup, elegida mediante el hash de (valor + clave). La misma entrada selecciona siempre la misma línea, lo que da consistencia referencial entre tablas sin necesitar base de mapeo. Es el algoritmo de lookup más usado del plugin.

FORMATO DE ENTRADA

> Cualquier cadena no nula. El contenido del archivo determina el dominio de la salida.

ENTRADA → SALIDA

ENTRADA		SALIDA
`Juan Pérez`	→	**`Rodrigo`**
`María Gómez`	→	**`Pedro`**
`Juan Pérez` `(ALL_UPPER)`	→	**`RODRIGO`**

La tercera fila muestra que `maskedValueCase` solo afecta a la capitalización — la línea seleccionada del archivo sigue siendo la misma.

PARÁMETROS

`lookupFile` *	Archivo con un valor sustituto por línea. El número de líneas define la diversidad de la salida: un archivo de 20 nombres hará que muchos valores distintos colisionen en el mismo sustituto.
`hashMethod`	Cómo se deriva el índice de la línea. **SHA256** — hash criptográfico, determinista por la clave (recomendado). **LEGACY** — el método antiguo, mantenido por compatibilidad con datos ya enmascarados. **RANDOMIZE** — selección aleatoria, *no* determinista: cada ejecución cambia el resultado.
`maskedValueCase`	Capitalización de la salida. **PRESERVE_LOOKUP_FILE** — como en el archivo. **PRESERVE_INPUT** — copia el patrón de la entrada. **ALL_LOWER** / **ALL_UPPER**.
`inputCaseSensitive`	Cuando es verdadero, "Juan" y "JUAN" producen hashes distintos y pueden recibir sustitutos diferentes. Déjalo en falso para tratar las variaciones de mayúsculas como el mismo valor.
`trimWhitespaceFromInput`	Elimina los espacios de ambos extremos de la entrada antes del hash. Hace que `"  Juan  "` y `"Juan"` caigan en el mismo sustituto — importante en columnas `CHAR(n)` .
`trimWhitespaceInLookupFile`	Elimina los espacios de ambos extremos de cada línea del archivo al cargarlo, evitando que las líneas con espacios sobrantes se conviertan en valores distintos.

7.2 Null-Safe Secure Lookup

`algorithm.plugin.nullSecureLookup.NullSecureLookup` **LIMITADO EN EL RUNNER**

Variante de Secure Lookup con tratamiento explícito de valores nulos: una entrada nula devuelve nulo en lugar de lanzar error. No expone parámetros — usa el archivo de lookup incorporado en el plugin.

FORMATO DE ENTRADA

> Cualquier cadena, o nulo.

ENTRADA → SALIDA

ENTRADA		SALIDA
`valor-sensible`	→	`null` (en el runner independiente)

PARÁMETROS

Ninguno.

> **En el runner independiente siempre devuelve `null`.** El algoritmo depende del contexto del Masking Engine para resolver su archivo de lookup incorporado. Un resultado `null` aquí es lo esperado y no indica un error de configuración.

7.3 Data Cleansing

`algorithm.plugin.dataCleansing.DataCleansing` **REQUIERE ARCHIVO**

Sustituye valores según una tabla de equivalencias **explícita**: tú declaras exactamente qué valor pasa a cuál. A diferencia de Secure Lookup, no hay hash ni sorteo. En la práctica es más una herramienta de estandarización que de enmascaramiento — normalizar siglas, unificar grafías divergentes.

FORMATO DE ENTRADA

> Cadena con correspondencia en el archivo. Sin correspondencia, el valor original pasa **sin alterar**.

ENTRADA → SALIDA

ENTRADA		SALIDA
MAD	→	**Madrid**
bcn	→	**Barcelona**
XX	→	**XX** (sin correspondencia)

La segunda fila coincidió pese a la diferencia de mayúsculas porque `caseSensitive` está en falso.

PARÁMETROS

lookupFile *	Archivo con un mapeo por línea, en formato `original{delimitador}sustituto`. Ej.: `MAD,Madrid`.
delimiter	Separador entre original y sustituto. Ej.: `,` para CSV, `;`, o tabulación. Omitido, usa tabulación.
caseSensitive	Cuando es verdadero, `SP` y `sp` necesitan entradas separadas en el archivo. Falso es lo más práctico en la mayoría de los casos.
trimWhitespace	Elimina los espacios de ambos extremos de la entrada y de las líneas del archivo antes de comparar, evitando fallos de correspondencia por el relleno de la base de datos.

> **Los valores sin correspondencia pasan intactos.** Esto significa que una tabla de equivalencias incompleta deja escapar datos originales **silenciosamente**. Si la columna es sensible, valida la cobertura del archivo antes de ejecutar en producción.

7.4 Mapping

`algorithm.plugin.mapping.Mapping` **NO FUNCIONA EN EL RUNNER**

Mantiene un mapeo persistente uno a uno en base de datos: cada valor original recibe un sustituto único, guardado y reutilizado en todas las tablas y ejecuciones futuras. Es la garantía más fuerte de **consistencia referencial** del plugin — y la única que sobrevive entre jobs distintos.

FORMATO DE ENTRADA

> Cualquier cadena. El algoritmo consulta el mapping set y, si el valor es nuevo, crea y persiste un sustituto.

ENTRADA → SALIDA

ENTRADA		SALIDA
`cliente@email.com`	→	`Error: Mapping set references are not supported`

PARÁMETROS

`mappingSet` *	Referencia al conjunto de mapeo que almacena las correspondencias.
↳ **`algorithmName`** *	Nombre del mapping set en el Masking Engine — identifica qué conjunto usar. Ej.: `client-name-mapping`.
↳ **`host`**	Host de la base de datos que aloja el mapping set, cuando es remoto.
↳ **`port`**	Puerto de la base de datos remota del mapping set.
↳ **`database`**	Nombre de la base de datos remota.
↳ **`schema`**	Esquema de la base de datos remota.
↳ **`isRemote`**	Cuando es verdadero, usa los datos de conexión anteriores. Falso usa la instancia local del Engine.
↳ **`mappingLookupKey`**	Clave de partición que permite que el mismo mapping set sirva a múltiples contextos aislados — por cliente, por entorno.
↳ **`propertiesRef`**	Archivo de propiedades con la configuración de conexión, como alternativa a rellenar host/puerto/base/esquema individualmente.
`ignoreCharacters`	Códigos ASCII a eliminar de la entrada antes de la consulta. Ej.: `45` guion, `46` punto — haciendo que un CPF con y sin formato apunte al mismo mapeo.

> **No disponible en el Algorithm Tester.** El runner independiente no tiene base de mapeo; cualquier prueba devuelve `Mapping set references are not supported`. Este algoritmo solo puede validarse en el Masking Engine.

PARTE 8

Multi-Column

Algoritmos que reciben la fila entera en lugar de un valor aislado — permitiendo que el enmascaramiento de una columna dependa del contenido de otra.

8.1 Multi-Column Condition

`algorithm.plugin.conditional.MultiColumnCondition` **DETERMINISTA** **MULTICOLUMNA**

Elige qué algoritmo aplicar a cada columna **según el valor de una columna condicional**. El caso clásico: si la columna de género dice "F", enmascarar el nombre con una lista de nombres femeninos; si dice "M", con nombres masculinos — manteniendo la coherencia interna del registro.

FORMATO DE ENTRADA

> Una fila con columnas nombradas. Obligatoria: `key` (la columna condicional). Opcionales: `string1` … `string10`, `numeric1` … `numeric3`, `date1` … `date3`, `binary1` … `binary3`.

ENTRADA → SALIDA

FILA DE ENTRADA		FILA DE SALIDA
`key = "F"` `string1 = "María"`	→	**`key = "F"`** **`string1 = "Jalisa"`**

La columna `key` no se enmascara — solo decide qué condición aplica. Únicamente `string1` tenía algoritmo configurado.

PARÁMETROS

`conditions`	Lista de condiciones evaluadas contra el valor de `key`. Cada condición declara los valores que la activan y qué algoritmos aplicar a qué ranuras de columna.
↳ **`key`**	Lista de valores de la columna condicional que activan esta condición. Ej.: `["F", "Female"]` se activa con cualquiera de los dos.
↳ **`string1…string10`**	Algoritmo aplicado a la columna `stringN` cuando la condición se activa. Ej.: `{"name": "dlpx-core:FirstName"}`. Las ranuras sin algoritmo pasan intactas.
↳ **`numeric1…numeric3`**	Algoritmo aplicado a las columnas numéricas (`BigDecimal`).
↳ **`date1…date3`**	Algoritmo aplicado a las columnas de fecha (`LocalDateTime`).
↳ **`binary1…binary3`**	Algoritmo aplicado a las columnas binarias (`ByteBuffer`).
`fallbackAlgo`	Algoritmo aplicado a las columnas string cuando *ninguna* condición coincide. Sin él, los valores pasan sin enmascarar.
`fallbackKey`	Valor asumido como clave cuando la columna `key` es nula o falta en la fila.
`keyCaseSensitive`	Cuando es verdadero, la comparación de `key` con las listas de las condiciones distingue mayúsculas de minúsculas. Por defecto: falso.
`filterLength`	Usa solo los primeros (o últimos) *N* caracteres de `key` en la comparación. Útil cuando la columna condicional tiene un prefijo significativo y un sufijo variable.
`filterDirection`	Desde dónde contar los caracteres de `filterLength`: `LEFT` (el inicio) o `RIGHT` (el final).

> **De dónde salen los nombres `key`, `string1` …** Son **ranuras fijas** del algoritmo, no nombres de columnas de tu base de datos. La traducción entre la columna real (`GENERO`, `NOMBRE`) y la ranura (`key`, `string1`) se hace en el **inventario** del Masking Engine. En el Algorithm Tester simulas ese mapeo rellenando la tabla de entrada a mano.

> **Las columnas sin algoritmo no se enmascaran.** Si una condición declara solo `string1`, todas las demás columnas de la fila pasan intactas — aunque contengan datos sensibles. Configura `fallbackAlgo` o declara cada ranura explícitamente.

8.2 Multi-Column Address

`algorithm.plugin.address.MultiColumnAddress` **REQUIERE ARCHIVO DE DIRECCIONES**

Enmascara campos de dirección repartidos en varias columnas — calle, número, barrio, ciudad, provincia, código postal — generando una dirección sintética **coherente entre sí**. Sin él, enmascarar cada columna por separado produciría combinaciones imposibles (una calle de Madrid con un código postal de Barcelona).

FORMATO DE ENTRADA

> Las columnas de dirección de la fila. Requiere un archivo de lookup de direcciones en el formato específico de Delphix.

ENTRADA → SALIDA

ENTRADA		SALIDA
`Calle de las Flores, 123`	→	`Error: FileReference nulo — archivo de direcciones no configurado`

PARÁMETROS

No documentados en esta guía — el algoritmo no se inicializa sin el archivo de direcciones, lo que impide inspeccionar su esquema en el runner.

> **No comprobable en el runner independiente.** El algoritmo falla durante la inicialización sin el archivo de lookup de direcciones, que no viene con el plugin. Configúralo y valida directamente en el Masking Engine.

PARTE 9

Otros

Dos algoritmos especializados: recálculo genérico de dígito verificador y tokenización reversible.

9.1 Check Digit

`algorithm.plugin.checkdigit.Checkdigit` **DETERMINISTA**

Enmascara números que llevan dígito verificador y **recalcula ese dígito** después, para que el resultado siga pasando la validación. Es genérico: cualquier esquema de suma ponderada más módulo puede describirse con sus parámetros — código de barras, EAN, recibo bancario, matrícula, inscripción fiscal.

FORMATO DE ENTRADA

> Cadena numérica sin puntuación, incluyendo el dígito verificador en la posición configurada. Ej.: `7891000315507` (EAN-13).

ENTRADA → SALIDA

ENTRADA		SALIDA
123456789012	→	**031477056818**

PARÁMETROS

weightList *	Pesos aplicados a cada dígito en la suma ponderada, uno por dígito de datos. Ej.: EAN-13 usa `[1,3,1,3,1,3,1,3,1,3,1,3]`. La longitud de la lista debe coincidir con `numDigitsForCheckdigitCalculation`.
modulusNumber *	Divisor del módulo. El verificador suele ser `(módulo − resto) % módulo`. EAN-13 usa `10`; CNPJ usa `11`.
checkDigitIndex *	Posición del dígito verificador en la cadena, contada desde `0` por la izquierda. En un código de 13 dígitos con el verificador al final, indica `12`.
calculateChecksumRightToLeft *	Dirección en que se aplican los pesos. **true** — de derecha a izquierda (recibos bancarios, CNPJ). **false** — de izquierda a derecha (EAN, UPC).
numDigitsForCheckdigitCalculation *	Cuántos dígitos entran en la suma, excluyendo el propio verificador. EAN-13 tiene 12 dígitos de datos.
swapModulusForZeroRemainder	Cuando el resto es cero, usa el propio `modulusNumber` como verificador en lugar de `0`. Algunos estándares (el CNPJ entre ellos) siguen esta regla.
checksumCalculationType	**STANDARD** — suma ponderada módulo N (por defecto). **TFN** — la variante australiana (Tax File Number), con lógica propia.
numericAlgorithm	Algoritmo que mezcla los dígitos que no son el verificador. Omitido, usa el Character Mapping numérico integrado.
alphaNumericAlgorithm	Algoritmo usado cuando la entrada contiene letras además de dígitos. Sin él, la entrada alfanumérica puede fallar según `characterHandling`.
fallbackAlgorithm	Algoritmo invocado con entrada inválida cuando `invalidInputHandling = FALLBACK_MASK`. Recibe el valor original entero.
preserveRegex	Regex que identifica partes a preservar — prefijos fijos, códigos de país. Los tramos que coincidan quedan intactos y solo se enmascara el resto.
inputHandlingConfig	Bloque que controla el tratamiento de la entrada antes del enmascaramiento.
↳ **characterHandling**	**NUMERIC_ONLY** — acepta solo `0–9`. **STANDARD** — las letras usan su valor ASCII. **ASCII_VALUE_MINUS_48** — las letras usan `(ASCII − 48)`, empleado por estándares que intercalan letras y dígitos.
↳ **invalidInputHandling**	**ERROR** — lanza excepción (por defecto). **FALLBACK_MASK** — delega en `fallbackAlgorithm`.

↳ **`shortInputHandling`**	Entrada más corta de lo esperado: **FALLBACK** delega, **PAD_LEFT** / **PAD_RIGHT** rellenan con `padCharacter`.
↳ **`padCharacter`**	Carácter de relleno para entradas cortas. Normalmente `0`.
↳ **`trimWhitespace`**	Elimina los espacios de ambos extremos antes de procesar. Necesario en columnas `CHAR(n)` con relleno.

9.2 Tokenization

`algorithm.plugin.tokenization.Tokenization` **REVERSIBLE** **⚠ NO DETERMINISTA POR DEFECTO**

El único algoritmo **reversible** del conjunto. Cifra el valor con AES y devuelve un token; con la misma clave y configuración, el valor original puede recuperarse (en el Tester, con el botón *Detokenizar* — modo `REIDENTIFY`). Es la elección cuando el dato debe volver a su forma original en algún punto del flujo.

FORMATO DE ENTRADA

> Cadena alfanumérica. Los caracteres fuera del conjunto tokenizable — símbolos, espacios, guiones — activan el `fallback`.

ENTRADA → SALIDA

ENTRADA		SALIDA
`4111111111111111`	→	**`aOyyO06W5exniSZq90F2ejgN1pWMbfrn`**
`aOyyO06W5exniSZq90F2ejgN1pWMbfrn` (REIDENTIFY)	→	**`4111111111111111`**
`4111-1111-1111-1111` (con guiones)	→	**`3P9ecws17K00ZE0bsMm1AKVTqMkbEabFgrFy`**

La segunda fila demuestra la reversión exacta. Nótese que el token es **más largo** que el original — lleva incorporado el vector de inicialización.

PARÁMETROS

`fallback`	Qué hacer con los caracteres no tokenizables. **NONE** — lanza error. **CHARACTER_MAPPING** — aplica enmascaramiento genérico a esos caracteres y continúa.
`ivLength`	Tamaño del vector de inicialización AES en bytes (por defecto `8`). **Es este parámetro el que decide si el algoritmo es determinista.** Con cualquier valor mayor que cero se sortea un IV nuevo en cada ejecución: la misma entrada con la misma clave genera un token distinto cada vez. Con `0` no hay IV y el token es siempre el mismo. Los valores mayores aumentan la aleatoriedad, pero consumen más espacio en el resultado.
`cmCharacterGroups`	Solo con `fallback = CHARACTER_MAPPING`. Define qué caracteres son intercambiables en el fallback. Vacío usa los grupos por defecto.
`cmMinMaskedPositions`	Solo con `fallback = CHARACTER_MAPPING`. Mínimo de posiciones que el fallback debe enmascarar, evitando tokens en los que casi nada cambió.

> ⚠ **No determinista con la configuración por defecto.** Con `ivLength` mayor que cero — y el valor por defecto es `8` — cada ejecución sortea un vector de inicialización nuevo, así que la misma entrada con la misma clave produce un token distinto cada vez. Todos se revierten correctamente, pero **no sirven para preservar joins**: el mismo valor en dos tablas se convierte en dos tokens distintos. Si necesitas que el mismo valor genere siempre el mismo token, usa `ivLength: 0` — la salida es entonces determinista y sigue siendo reversible.

> **El token no preserva el formato.** Pese a describirse como *format-preserving*, el resultado observado es una cadena Base64 más larga que la entrada — `4111111111111111` (16 caracteres) pasó a 32. Dimensiona la columna destino con holgura, o el job fallará por truncamiento.

> **La clave es el secreto.** Quien tenga la clave puede revertir cualquier token. Trátala con el mismo rigor que una clave de producción: fuera del código, fuera del repositorio, con rotación controlada — y recuerda que rotar invalida todos los tokens ya emitidos.

APÉNDICE A

Conceptos transversales

Comportamientos que atraviesan varios algoritmos y suelen causar confusión la primera vez que se encuentran.

DETERMINISMO Y EL PAPEL DE LA CLAVE

La mayoría de los algoritmos es **determinista**: el mismo valor de entrada, con la misma clave, produce siempre la misma salida. Eso es lo que preserva las uniones — si `Juan` pasa a `Mariana` en la tabla de clientes, pasa a `Mariana` también en la de pedidos.

La clave es lo que hace el resultado impredecible para quien no la posee. Cambiarla cambia toda la salida, como se vio en §4.1. Dos consecuencias prácticas: **debe usarse la misma clave en todo el proyecto**, y **cambiar la clave invalida la consistencia con datos ya enmascarados**.

COMPORTAMIENTO	ALGORITMOS
Determinista por clave	Character Mapping, Numeric Mapping, Segment Mapping, Payment Card, IBAN, Check Digit, Date Shift, Date Replacement, Secure Lookup, Name, Full Name, Email, Phone, Financial ID BR
⚠ **No determinista**	**Shuffle** (§4.6) — siempre; permutación distinta en cada ejecución. **Tokenization** (§9.2) — con `ivLength` mayor que cero, y el valor por defecto es `8`; usa `ivLength: 0` para salida determinista. **Secure Lookup** (§7.1) — solo con `hashMethod: RANDOMIZE`.
Independiente de la clave	Min/Max (ambos), Redact, Free Text Redaction, Data Cleansing, Repeat First Digit, Numeric Expression sin `seed`

ALGORITMOS QUE PUEDEN NO ENMASCARAR NADA

Varios algoritmos devuelven el valor original en silencio bajo ciertas condiciones. Ninguno emite advertencia — la verificación corre por tu cuenta.

ALGORITMO	CUÁNDO EL DATO PASA INTACTO
Min/Max BigDecimal · Date/Time	Siempre que el valor ya esté dentro del rango
Data Cleansing	Valor sin correspondencia en el archivo
Redact	`regexRedact` vacío, o ningún patrón coincidente
Regex Decompose	`fallbackAction: PRESERVE` con un valor que no coincide con ningún patrón
Full Name	`firstNameAlgorithmRef` o `lastNameAlgorithmRef` sin configurar
Multi-Column Condition	Ranura de columna sin algoritmo, o ninguna condición coincidente sin `fallbackAlgo`
Character Mapping	Caracteres fuera de todos los `characterGroups` (acentos, puntuación)

COLUMNAS CHAR(N) Y EL RELLENO INVISIBLE

Las columnas `CHAR(n)` rellenan el valor con espacios a la derecha hasta el tamaño declarado. Un CPF de 11 dígitos en una columna `CHAR(15)` llega al algoritmo como `"12345678909    "`. Como `\d` no coincide con un espacio, las regex que funcionan en el Tester fallan en producción con mensajes como `No pattern matched` o `did not conform to the expected format`.

La solución es activar la opción de recorte del algoritmo: `trimInput` en Regex Decompose, `trimWhitespace` en Check Digit y Data Cleansing, `trimWhitespaceFromInput` en Secure Lookup. El

relleno se restaura en la salida.

REFERENCIAS ENTRE ALGORITMOS

Muchos parámetros piden una "referencia a algoritmo" — `firstNameAlgorithmRef`, `actions.algorithm`, `numericAlgorithm`, `fallbackAlgo`. El valor esperado es `{"name": "<instancia>"}`, donde la instancia puede ser:

① un nombre del catálogo integrado, con el prefijo `dlpx-core:` (apéndice B); ② el nombre de una prueba guardada en el propio Algorithm Tester; ③ en el Masking Engine, cualquier instancia de algoritmo configurada. Los nombres inexistentes producen `Sub-algorithm not found`.

APÉNDICE B

Catálogo de instancias integradas

Las 62 instancias que acompañan al plugin, utilizables en cualquier parámetro de referencia. Anteponles `dlpx-core:` — por ejemplo `dlpx-core:FirstName`. Los nombres con espacios funcionan sin problema.

- ABARoutingNumber SL
- Age SL
- BR - Financial ID
- BloodType SL
- Brazil CNPJ
- Brazil CPF
- CM Alpha-Numeric
- CM Digits
- CM Numeric
- Countries SL
- Credit Card
- Date Shift Discrete
- Date Shift Fixed
- Date Shift Variable
- Department SL
- Email SL
- Email Unique
- Ethnicity SL
- FirstName
- FullName
- Gender SL
- IBAN
- Japan Corporate Number
- Japan Drivers License
- Japan My Number
- JobTitle SL
- Language SL
- LastName
- Lat_Long Coordinates
- Legacy AccNoLookup
- Legacy AddrLine2Lookup
- Legacy AddrLookup
- Legacy BusinessLegalEntityLookup
- Legacy CommentLookup
- Legacy DrivingLicenseNoLookup
- Legacy DummyHospitalNameLookup
- Legacy EmailLookup
- Legacy FirstNameLookup
- Legacy FullNMLookup
- Legacy LastCommaFirstLookup
- Legacy LastNameLookup
- Legacy RandomValueLookup
- Legacy SchoolNameLookup
- Legacy USStateCodesLookup
- Legacy USStatesLookup
- Legacy UsCitiesLookup
- Legacy UsCountiesLookup
- Legacy WebURLsLookup
- MaritalStatus SL
- MedicalGenericDrug SL
- NationalOrigin SL
- Null Secure Lookup
- Phone US
- Phone Unique
- PoliticalOrientation SL
- Redact Digits-Zero
- ReligiousOrientation SL
- Repeat First Digit
- SexualOrientation SL
- Shuffle
- SwiftCode SL
- TimeRange

> **El sufijo `SL`** significa *Secure Lookup* — instancias preconfiguradas con archivos de lookup temáticos incorporados (profesiones, países, estado civil). Son la forma más rápida de enmascarar columnas categóricas sin montar un archivo propio.

APÉNDICE C

Limitaciones del runner independiente

El Algorithm Tester ejecuta los algoritmos fuera del Masking Engine. Algunas funciones dependen del contexto del Engine y no funcionan en local.

FUNCIÓN	SITUACIÓN
`FileReference`	**Compatible.** Usa una URI `file:///ruta/absoluta`. Requiere `commons-codec` en `lib/`. Los archivos se gestionan en *Configuración → Archivos*.
`MappingSetReference`	**No compatible.** El algoritmo *Mapping* (§7.4) falla con `Mapping set references are not supported` — depende de una base de mapeo.
`EmbeddedGeneratorReference`	**No compatible.**
`MaskValueMetadata`	Devuelve siempre `null`.
Multi-Column Address (§8.2)	No se inicializa sin el archivo de lookup de direcciones de Delphix.
Null-Safe Secure Lookup (§7.2)	Devuelve siempre `null` — depende del archivo incorporado que resuelve el Engine.

> **Sobre las salidas de esta guía.** Todos los pares entrada → salida se generaron ejecutando los algoritmos en `AlgorithmRunner` con el plugin `delphix-algorithm-plugin 2026.3.0-SNAPSHOT`. Los algoritmos deterministas reproducen exactamente estos valores con la misma clave; Shuffle, por su naturaleza, produce una permutación distinta en cada ejecución.