DELPHIX MASKING · REFERÊNCIA TÉCNICA

Guia dos Algoritmos de Mascaramento

Como cada algoritmo transforma o dado, com que parâmetros, e o que esperar na saída.

31	9	62
ALGORITMOS	CATEGORIAS	INSTÂNCIAS BUILT-IN

Plugin `delphix-algorithm-plugin 2026.3.0`

Saídas verificadas no AlgorithmRunner

Índice

Os algoritmos estão agrupados pelas mesmas categorias usadas na barra lateral do Algorithm Tester. Todos os pares entrada → saída deste guia foram gerados executando o algoritmo de verdade — não são ilustrativos.

PARTE 1

Dados Pessoais

Algoritmos que reconhecem a semântica do campo — e-mail, telefone, nome, CPF/CNPJ — e geram substitutos sintéticos que continuam parecendo (e validando como) dados reais.

1.1 Email

`algorithm.plugin.email.Email` **DETERMINÍSTICO**

Mascara um endereço de e-mail tratando as duas metades separadamente: a parte antes do `@` (o nome) e o domínio depois dele. Cada metade tem sua própria estratégia, o que permite, por exemplo, anonimizar completamente o usuário mas preservar o domínio para manter a distribuição por empresa nos relatórios.

FORMATO DE ENTRADA

> Endereço de e-mail válido, com `@` e domínio. Ex.: `joao.silva@empresa.com.br`. Entradas fora do formato caem no `errorHandlingAction`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`joao.silva@empresa.com.br`	→	**`TZW6C7VX6U4EOAWDXVGCD3XIZFNA7QGP2C4AFMSE3ANRQXHTVLGA@example.com`**
`joao.silva@empresa.com.br`	→	**`TZW6C7VX6U4EOAWDXVGCD3XIZFNA7QGP2C4AFMSE3ANRQXHTVLGA@empresa.com.br`**

Primeira linha com `domainAction: REPLACEMENT`; segunda com `domainAction: PRESERVE`. Note que o nome gerado é idêntico nos dois casos — ele depende só do input e da chave.

PARÂMETROS

`nameAction` *	Como mascarar a parte do nome. **UNIQUE** — gera um identificador longo e único por input (garante que dois e-mails diferentes nunca colidam). **LOOKUP** — sorteia um nome de um arquivo de lookup. **APPLY_ALGORITHM** — passa o nome por outro algoritmo. **APPLY_FIRSTNAME_AND_LASTNAME_ALGORITHMS** — divide o nome e aplica algoritmos de primeiro nome e sobrenome separadamente, gerando algo como `ana.silva@…`.
`domainAction` *	Como tratar o domínio. **REPLACEMENT** — troca por `domainReplacementString`. **APPLY_ALGORITHM** — passa o domínio por outro algoritmo. **PRESERVE** — mantém o domínio original intacto.
`domainReplacementString`	O domínio substituto, usado apenas quando `domainAction = REPLACEMENT`. Ex.: `example.com`.
`errorHandlingAction`	O que fazer quando a entrada não é um e-mail válido. **EXCEPTION** — interrompe com erro. **CHARACTER_MAPPING** — aplica um mascaramento genérico caractere a caractere. **PRESERVE** — devolve o valor original sem alterar.

> **Quando usar qual.** Se o objetivo é apenas impedir o envio acidental de e-mails, `domainAction: REPLACEMENT` para um domínio que você controla é o mais seguro. Se analistas precisam continuar segmentando por empresa, use `PRESERVE` — mas lembre que o domínio pode ser identificador em bases pequenas.

1.2 Phone

`algorithm.plugin.phone.Phone` **DETERMINÍSTICO**

Gera um número de telefone sintético preservando exatamente o formato do original: parênteses, espaços, hífens e a quantidade de dígitos permanecem no lugar. Apenas os dígitos mudam — e nem todos: o algoritmo tende a preservar prefixos que identificam o formato (DDD, o `9` inicial de celular).

FORMATO DE ENTRADA

> Telefone em qualquer formatação. A pontuação é mantida posicionalmente na saída. Ex.: `(11) 99999-8888`, `11987654321`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`(11) 99999-8888`	→	**`(11) 99468-0822`**
`11987654321`	→	**`11984152108`**

PARÂMETROS

`isPhoneUnique` *	Quando verdadeiro, o algoritmo garante que dois telefones de origem diferentes nunca produzam o mesmo número mascarado. Isso é importante quando o telefone é usado como chave de deduplicação ou junção. Quando falso, colisões são possíveis, mas os números gerados ficam mais "naturais".

1.3 Name

`algorithm.plugin.name.Name` **DETERMINÍSTICO** **REQUER ARQUIVO**

Substitui um nome simples — só o primeiro nome, ou só o sobrenome — por outro sorteado de um arquivo de lookup. O sorteio é um hash de (valor + chave), então o mesmo nome sempre vira o mesmo substituto: "João" será sempre "Ana" em todas as tabelas do projeto, o que preserva junções por nome. Para nomes compostos, use o **Full Name** (§1.4).

FORMATO DE ENTRADA

> Uma única palavra, sem separadores. Ex.: `João`, `Silva`, `Maria`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
João	→	**Ana**
MARIA	→	**LUCAS**

Com `maskedValueCase: PRESERVE_INPUT` — a entrada em caixa alta produziu saída em caixa alta, mesmo o arquivo contendo "Lucas".

PARÂMETROS

`lookupFile` *	Arquivo de texto com um nome por linha. É a fonte dos substitutos. Quanto mais linhas, menor a chance de dois nomes diferentes receberem o mesmo substituto. Pode conter qualquer lista — primeiros nomes, sobrenomes, nomes masculinos ou femininos separadamente.
`maskedValueCase`	Capitalização do resultado. **PRESERVE_LOOKUP_FILE** — exatamente como está no arquivo. **PRESERVE_INPUT** — copia o padrão de caixa do input. **ALL_LOWER** / **ALL_UPPER** — força minúsculas ou maiúsculas.
`particlesToRemoveFile`	Arquivo com partículas (uma por linha) removidas do input *antes* de calcular o hash. Serve para que "da Silva" e "Silva" produzam o mesmo substituto, em vez de tratá-los como nomes distintos.
`particlesToPreserveFile`	Arquivo com partículas que devem reaparecer no resultado. O substituto gerado recebe de volta as partículas que o input original tinha.
`maxLengthOfMaskedName`	Comprimento máximo em caracteres. Nomes do arquivo que excedem o limite são descartados no sorteio. Use quando a coluna de destino tem restrição de tamanho e você não quer truncamento.
`maxNumberNames`	Máximo de nomes retornados em modo multi-valor. Para uma coluna simples, deixe vazio.
`filterAccent`	Quando verdadeiro, remove acentos antes de calcular o hash — fazendo "José" e "Jose" produzirem o mesmo substituto. Recomendado em bases brasileiras onde a acentuação é inconsistente.
`inputCaseSensitive`	Quando verdadeiro, "João" e "JOÃO" são inputs distintos e podem receber substitutos diferentes. O padrão (falso) ignora a caixa ao comparar.

1.4 Full Name

`algorithm.plugin.name.FullName` **DETERMINÍSTICO**

Quebra um nome completo em primeiro(s) nome(s) e sobrenome, e aplica um algoritmo diferente em cada parte. É um orquestrador: ele não mascara nada sozinho, apenas decide onde termina o nome e começa o sobrenome, e delega cada pedaço ao algoritmo que você indicar.

FORMATO DE ENTRADA

> Nome completo separado por espaços. Ex.: `João da Silva`, `Maria Oliveira Santos`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
João da Silva Santos	→	**Vyvind zouhair Vesna**

PARÂMETROS

lastNameAtTheEnd	Onde está o sobrenome. **true** — é a última palavra (padrão brasileiro: "João da Silva" → sobrenome "Silva"). **false** — é a primeira (padrão de bases que gravam "Silva, João").
ifSingleWordConsiderAsLastName	Desempate para input de uma palavra só. **true** — trata como sobrenome. **false** — trata como primeiro nome. Define qual dos dois algoritmos será chamado.
firstNameAlgorithmRef	Algoritmo aplicado ao(s) primeiro(s) nome(s). Ex.: `dlpx-core:FirstName`. Se omitido, os primeiros nomes ficam *sem mascaramento*.
lastNameAlgorithmRef	Algoritmo aplicado ao sobrenome. Ex.: `dlpx-core:LastName`. Se omitido, o sobrenome fica sem mascaramento.
lastNameSeparators	Separadores adicionais além do espaço. Ex.: `["-"]` faz "Maria-Silva" ser lido como nome "Maria" + sobrenome "Silva".
maxLengthOfMaskedName	Limite de caracteres do nome completo mascarado, para respeitar a largura da coluna de destino.
maxNumberFirstNames	Quantos primeiros nomes mascarar. Com valor `1`, "João Pedro Carlos Silva" tem só "João" mascarado — "Pedro Carlos" passa intacto.

> **Atenção aos padrões.** Os dois `…AlgorithmRef` são opcionais, e quando ficam vazios a parte correspondente **não é mascarada**. É fácil configurar só o sobrenome e deixar os primeiros nomes vazando em produção — confira sempre os dois.

1.5 Financial ID BR (CPF/CNPJ)

`algorithm.plugin.financialId.FinancialIdBr` **DETERMINÍSTICO**

Mascara CPF e CNPJ gerando números sintéticos que continuam **matematicamente válidos** — os dígitos verificadores são recalculados. O tipo é detectado automaticamente pela quantidade de dígitos: 11 é CPF, 14 é CNPJ. A formatação de entrada (pontos, barra, hífen) é preservada na saída.

FORMATO DE ENTRADA

> CPF ou CNPJ, com ou sem formatação. Ex.: `123.456.789-09`, `12345678909`, `11.222.333/0001-81`, `11222333000181`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`123.456.789-09`	→	**`210.747.284-08`**
`11.222.333/0001-81`	→	**`60.628.464/0001-79`**
`11222333000181`	→	**`60628464000179`**
`12345678000190`	→	`Erro: dígitos verificadores inválidos (maskInvalidInput = false)`

As linhas 2 e 3 mostram o mesmo CNPJ com e sem pontuação: os dígitos gerados são idênticos, só o formato muda.

PARÂMETROS

`maskInvalidInput`	O que fazer quando o CPF/CNPJ de entrada não passa na validação dos dígitos verificadores. **false** (padrão) — lança erro e interrompe. **true** — mascara mesmo assim, ignorando a invalidade. Bases legadas costumam ter documentos inválidos gravados; se for o seu caso, ative isto ou configure o `fallbackAlgorithm`.
`cmNumericRef`	Algoritmo usado internamente para embaralhar os dígitos antes do recálculo do verificador. Se vazio, usa o Character Mapping numérico embutido. Informe uma instância do catálogo, ex.: `dlpx-core:CM Digits`.
`fallbackAlgorithm`	Algoritmo acionado quando o valor não é reconhecido como CPF nem CNPJ — campo em branco, texto livre, comprimento inesperado. Sem ele, esses casos viram erro.

PARTE 2

Datas

Quatro estratégias distintas para datas: deslocar preservando intervalos, deslocar por valores discretos, substituir por completo, ou apenas limitar a uma faixa.

2.1 Date Shift

`algorithm.plugin.dateAlgorithms.DateShift` **DETERMINÍSTICO**

Desloca a data por uma quantidade sorteada dentro de `[minRange, maxRange]`. O sorteio é determinístico pela chave, então a mesma data sempre recebe o mesmo deslocamento — o que **preserva a ordem cronológica e as distâncias** entre registros de uma mesma entidade. É o algoritmo de escolha quando análises dependem de intervalos ("dias entre pedido e entrega").

FORMATO DE ENTRADA

> ISO 8601: `yyyy-MM-dd` ou `yyyy-MM-ddTHH:mm:ss`. A saída sempre inclui a componente de hora.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
2024-03-15	→	**2024-08-09T00:00**
2024-03-15T14:30:00	→	**2024-12-17T14:30**
2024-01-31	→	**2024-05-31T00:00**

As duas primeiras com `±365 DAYS`; a terceira com `±12 MONTHS`. Note que a hora original é sempre preservada.

PARÂMETROS

minRange	Deslocamento mínimo, na unidade de `unit`. Negativo permite recuar no tempo. Ex.: `-365` com `DAYS` permite voltar até um ano.
maxRange	Deslocamento máximo. Positivo permite avançar. Um intervalo estreito preserva melhor a plausibilidade dos dados; um intervalo largo aumenta a proteção.
unit	Unidade do deslocamento: `SECONDS`, `MINUTES`, `HOURS`, `DAYS`, `MONTHS`, `YEARS`.
roll	Só importa com `MONTHS` ou `YEARS`, quando o resultado cairia em data inexistente (31 de janeiro + 1 mês = 31 de fevereiro). **true** — encolhe para o último dia válido do mês (28/fev). **false** — transborda para o mês seguinte (3/mar). Na dúvida, `false`.

> **Preservação de intervalos.** Como o deslocamento depende da data de origem, dois registros com datas diferentes recebem deslocamentos diferentes — a distância entre eles **não** é preservada exatamente. Se o requisito for manter distâncias intactas, o deslocamento precisa ser fixo por entidade, o que se obtém com `dlpx-core:Date Shift Fixed`.

2.2 Date Shift Discrete

`algorithm.plugin.dateAlgorithms.DateShiftDiscrete` **DETERMINÍSTICO**

Variante do Date Shift em que o deslocamento não vem de um intervalo contínuo, mas de um **conjunto finito de valores possíveis** definido em arquivo de configuração. Útil quando o negócio exige deslocamentos padronizados (por exemplo, apenas múltiplos de 7 dias, para preservar o dia da semana).

FORMATO DE ENTRADA

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

ENTRADA → SAÍDA

ENTRADA		SAÍDA
2024-03-15	→	**2024-03-07T00:00**

PARÂMETROS

Sem parâmetros expostos no schema — o conjunto de deslocamentos vem da configuração embutida do plugin.

2.3 Date Replacement

`algorithm.plugin.dateAlgorithms.DateReplacement` **DETERMINÍSTICO**

Descarta a data original e sorteia uma data **absoluta** dentro de `[minDate, maxDate]`. Diferente do Date Shift, aqui não há relação alguma entre entrada e saída além do determinismo — a ordem cronológica original é destruída. Use quando a data em si é sensível e nenhuma análise temporal depende dela.

FORMATO DE ENTRADA

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

ENTRADA → SAÍDA

ENTRADA		SAÍDA
1985-07-20T00:00:00	→	**1998-03-21T00:00**

PARÂMETROS

minDate	Data mínima possível na saída, no formato `yyyy-MM-ddTHH:mm:ss`. Define o piso da janela de sorteio.
maxDate	Data máxima possível, mesmo formato. Escolha uma janela plausível para o domínio — datas de nascimento em 1970–2000, por exemplo.
unit	Granularidade do sorteio: `SECONDS`, `MINUTES`, `HOURS` ou `DAYS`. Com `DAYS`, a componente de hora fica zerada.

2.4 Min/Max Date/Time

`algorithm.plugin.minMax.MinMaxLocalDateTime`

Não é bem um mascaramento, e sim um **truncamento de faixa** (clamping): datas dentro do intervalo passam intactas; datas fora são puxadas para o limite mais próximo. O uso típico é suprimir outliers que identificam indivíduos — o cliente centenário, a data de 1900 usada como sentinela.

FORMATO DE ENTRADA

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

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`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`**

Faixa `1970-01-01` a `2000-12-31T23:59:59`. A primeira data está dentro e passa inalterada; a segunda é anterior ao piso; a terceira é posterior ao teto.

PARÂMETROS

`minDate` *	Piso do intervalo. Qualquer data anterior é substituída por este valor.
`maxDate` *	Teto do intervalo. Qualquer data posterior é substituída por este valor.
`nonConformingDataDefaultValue`	Valor devolvido quando a entrada não é uma data/hora válida. Se ficar vazio, o algoritmo lança erro de dado não-conforme e interrompe o job.

> **Isto não anonimiza.** Valores dentro da faixa saem **idênticos** à entrada. Min/Max é uma ferramenta de supressão de outliers, não de mascaramento — nunca use sozinho em uma coluna sensível.

PARTE 3

Numérico

Transformações sobre valores numéricos: mapeamento determinístico para uma faixa, limitação de intervalo, expressões Java arbitrárias e uma regra fixa de repetição de dígitos.

3.1 Numeric Mapping

`algorithm.plugin.characterMapping.NumericMapping` **DETERMINÍSTICO**

Mapeia um número para outro dentro de `[minValue, maxValue]`, de forma determinística. Cada valor de origem recebe consistentemente o mesmo destino, o que preserva junções e contagens distintas — mas não a ordem nem a magnitude.

FORMATO DE ENTRADA

> Número inteiro positivo, sem pontuação nem casas decimais. Ex.: `12345`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
12345	→	**91472**
98765	→	**89966**
11111	→	**34588**

PARÂMETROS

`minValue`	Menor valor que a saída pode assumir. Escolha um piso que respeite a semântica da coluna (ex.: `10000` para garantir sempre 5 dígitos).
`maxValue`	Maior valor da saída. A largura da faixa determina a chance de colisão: faixas estreitas fazem valores distintos colidirem no mesmo destino.

3.2 Min/Max BigDecimal

`algorithm.plugin.minMax.MinMaxBigDecimal`

A contraparte numérica do Min/Max Date/Time: limita o valor a uma faixa, deixando passar intacto o que já está dentro dela. Serve para achatar outliers — salários muito altos, saldos negativos extremos — que sozinhos identificam a pessoa.

FORMATO DE ENTRADA

> Número decimal ou inteiro. Ex.: `5000.50`, `3.14`.

ENTRADA → SAÍDA

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

Faixa `1000` – `9999`. Note que a escala decimal do valor dentro da faixa é preservada.

PARÂMETROS

`minValue` *	Limite inferior. Valores menores são elevados a este número.
`maxValue` *	Limite superior. Valores maiores são rebaixados a este número.
`nonConformingDataDefaultValue`	Valor devolvido quando a entrada não é numérica. Vazio faz o algoritmo lançar erro de dado não-conforme.

> **Isto não anonimiza.** Assim como o Min/Max de datas, valores dentro da faixa saem idênticos. Combine com outro algoritmo se a coluna for sensível.

3.3 Numeric Expression

`algorithm.plugin.expression.NumericExpression`

O algoritmo mais aberto do conjunto: avalia uma **expressão Java de uma linha** sobre o valor de entrada. A expressão recebe duas variáveis — `input` (o valor como `BigDecimal`) e `seed` (um `long` derivado da chave, para mascaramento determinístico) — além de quaisquer constantes que você declarar.

FORMATO DE ENTRADA

> Número inteiro ou decimal. Ex.: `1000`, `3.14`.

ENTRADA → SAÍDA

ENTRADA · EXPRESSÃO		SAÍDA
`1234.56` `input.setScale(0, HALF_UP)`	→	**`1235`**
`99.4` `input.setScale(0, HALF_UP)`	→	**`99`**
`1234.56` `new BigDecimal(seed % 9000 + 1000)`	→	**`3701`**
`1000` `input.multiply(new BigDecimal(taxa))`	→	**`800.000000000000044408920985006261616945266723632 8125000`**

PARÂMETROS

`expression` *	Expressão Java que retorna `BigDecimal`. Variáveis disponíveis: `input` e `seed`. Classes precisam de nome totalmente qualificado — `new java.math.BigDecimal(...)`, `java.math.RoundingMode.HALF_UP`.
`inputType`	Como interpretar a entrada antes de expor em `input`: `DOUBLE`, `LONG` ou `BIG_DECIMAL` (padrão).
`constants`	Lista de constantes nomeadas acessíveis na expressão. Cada uma tem `name` e `value` (sempre string). Ex.: `name="taxa"`, `value="0.8"`, usada como `new java.math.BigDecimal(taxa)`.
`nonConformingDataDefaultValue`	Valor devolvido quando a entrada não é numérica. Vazio faz lançar erro.

> **Cuidado com ponto flutuante.** A quarta linha do exemplo devolveu `800.000000000000444…` em vez de `800`: a constante `"0.8"` passou por `double` antes de virar `BigDecimal`. Para aritmética exata, use o construtor de string — `new java.math.BigDecimal("0.8")` — e feche com `.setScale(2, java.math.RoundingMode.HALF_UP)`.

3.4 Repeat First Digit

`algorithm.plugin.repeatFirstDigit.RepeatFirstDigit`

Regra fixa e sem configuração: pega os **4 dígitos finais** do número e substitui todos pelo primeiro deles, repetido quatro vezes. O restante do número fica intacto. É um mascaramento fraco, adequado a campos de baixa sensibilidade onde só se quer quebrar o valor exato.

FORMATO DE ENTRADA

> String numérica com exatamente **4, 9, 10 ou 14 dígitos**, sem pontuação. Outros comprimentos são rejeitados.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
1234	→	**1111**
123456789	→	**123456666**

No segundo caso, os 5 primeiros dígitos são preservados e os 4 últimos (`6789`) viram `6666` .

PARÂMETROS

Nenhum. O comportamento é fixo.

PARTE 4

String

Os algoritmos de propósito geral para texto estruturado. Aqui estão as ferramentas mais usadas do plugin — e as mais configuráveis.

4.1 Character Mapping

`algorithm.plugin.characterMapping.CharacterMapping` **DETERMINÍSTICO**

Troca cada caractere por outro **do mesmo grupo**. Você define os grupos (letras minúsculas, maiúsculas, dígitos...) e o algoritmo garante que uma letra vire letra e um dígito vire dígito, preservando o comprimento e o "formato visual" do dado. Caracteres que não pertencem a nenhum grupo — espaços, acentos, pontuação — passam intactos.

FORMATO DE ENTRADA

> Qualquer string. Ex.: `João Silva`, `ABC123`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`João Silva123`	→	**`Qiãz Hfroy869`**
`ABC123`	→	**`BAG992`**
`Maria`	→	**`Rjewk`**
`João Silva123` `(outra chave)`	→	**`Fnãx Gsjgn414`**

O `ã` e o espaço sobreviveram — não estão em nenhum grupo configurado. A última linha mostra o efeito da chave: mesmo input, chave diferente, resultado diferente.

PARÂMETROS

`characterGroups`	Lista de strings; cada string é um conjunto de caracteres intercambiáveis entre si. Ex.: `["abcdefghijklmnopqrstuvwxyz", "0123456789"]` cria dois grupos isolados — letras só viram letras, dígitos só viram dígitos. Caracteres ausentes de todos os grupos nunca são mascarados.
`caseSensitive`	Quando verdadeiro, maiúsculas e minúsculas são tratadas como grupos separados, preservando a capitalização do original. Requer que você declare os dois conjuntos separadamente.
`minMaskedPositions`	Número mínimo de posições que precisam efetivamente mudar. Impede que o algoritmo devolva um valor quase idêntico ao original por coincidência do sorteio.
`preserveRanges`	Lista de trechos a manter intactos, cada um com `start`, `length` e `direction`. Serve para preservar prefixos ou sufixos com significado — código de agência, dígito de controle.
`preserveLeadingZeros`	Quando verdadeiro, zeros à esquerda são mantidos. Essencial em códigos numéricos gravados como texto, onde `007` e `7` não são equivalentes.

> **Por que os acentos não mudam.** Se caracteres acentuados precisam ser mascarados, inclua-os explicitamente em um grupo: `"aáàâãeéêiíoóôõuú"`. Do contrário eles ficam visíveis no dado mascarado e podem ajudar a reidentificar o registro.

4.2 Character Replacement

`algorithm.plugin.characterReplacement.CharacterReplacement`

Aplica regras de substituição caractere a caractere: cada regra declara *quais* caracteres detectar e *por qual* caractere trocá-los. Diferente do Character Mapping, aqui a substituição é fixa (não sorteada) — todos os caracteres filtrados viram o mesmo símbolo. Pode opcionalmente rodar depois de outro algoritmo, via `stringAlgorithm`.

FORMATO DE ENTRADA

> Qualquer string. Ex.: `Exemplo de texto`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`Exemplo de texto`	→	`*s*jnv* c* n*pd*`
`Maria Oliveira`	→	`H*q** *s****s*`
`João Coração` `(filterAccents: INPUT)`	→	`H*** Y***y**`

Regra: vogais → `*`. As consoantes também mudaram porque o algoritmo aplica um mascaramento próprio antes das regras. Na terceira linha, `filterAccents: INPUT` converteu `ã` e `ç` para ASCII antes de processar.

PARÂMETROS

`stringAlgorithm`	Algoritmo aplicado ao input *antes* das regras de substituição. O resultado dele é que passa pelas `replacementRules`. Permite compor: mascare com um algoritmo forte e depois normalize caracteres problemáticos.
`replacementRules`	Lista de regras aplicadas em sequência. Cada regra é um par (conjunto de caracteres a detectar, caractere substituto).
↳ **`filteredCharacters`**	Os caracteres que esta regra detecta. Forma simples, usada quando entrada e saída compartilham o mesmo conjunto.
↳ **`replacementCharacter`**	O caractere único que substitui cada ocorrência encontrada.
↳ **`inputFilteredCharacters`**	Forma avançada: caracteres detectados no input, quando você quer conjuntos diferentes para leitura e escrita.
↳ **`inputReplacementCharacter`**	Substituto aplicado às correspondências de `inputFilteredCharacters`.
↳ **`outputFilteredCharacters`**	Caracteres que, se aparecerem na *saída* do `stringAlgorithm`, também devem ser substituídos.
↳ **`outputReplacementCharacter`**	Substituto aplicado às correspondências de `outputFilteredCharacters`.
`requireInputChange`	Quando verdadeiro, exige que ao menos uma substituição tenha ocorrido — se nenhuma regra mudou nada, o algoritmo lança erro. Protege contra configurações que silenciosamente não mascaram nada.
`filterAccents`	Quando remover acentos. **INPUT** — antes de aplicar as regras. **OUTPUT** — no resultado final. **BOTH** — nas duas etapas. **NONE** — nunca (padrão).

4.3 Segment Mapping

`algorithm.plugin.segmentMapping.SegmentMapping` **DETERMINÍSTICO**

Divide a string em segmentos de comprimento fixo e trata cada um de forma independente — mascarar, preservar, ou substituir por constante. É o algoritmo certo para documentos de formato rígido onde partes têm significado diferente: CPF, CNPJ, CEP, código de barras, número de conta com agência embutida.

FORMATO DE ENTRADA

> String de formato fixo. Com `autoIgnoreCharacters` ativo, separadores (pontos, hífens, barras) são detectados e reposicionados automaticamente. Ex.: `123.456.789-09`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`123.456.789-09`	→	**`383.473.782-82`**
`987.654.321-00`	→	**`423.695.820-82`**

Segmentos de 3-3-3-2 dígitos, todos `MASK_NUMERIC`, com os separadores preservados nas posições originais.

PARÂMETROS

`segments` *	Lista ordenada de segmentos, na sequência em que aparecem no valor (desconsiderando caracteres ignorados). A soma dos comprimentos deve bater com o tamanho do dado.
↳ **`length`** *	Quantos caracteres este segmento ocupa. Não conta os caracteres ignorados.
↳ **`segmentType`** *	**MASK_NUMERIC** — troca por dígitos. **MASK_ALPHANUMERIC** — troca por caracteres alfanuméricos. **PRESERVE** — mantém o original. **CONSTANT** — substitui pelo valor fixo de `maskValues`.
↳ **`inputValues`**	Conjunto de caracteres aceitos neste segmento no input, como string contínua. Ex.: `"0123456789"`. Vazio aceita qualquer caractere.
↳ **`maskValues`**	Pool de caracteres usado na substituição. Ex.: `"123456789"` (sem o zero) impede que o segmento comece com `0`. Vazio usa o pool padrão do tipo.
`ignoreCharacters`	Códigos ASCII dos separadores a pular na contagem de posições — `46` ponto, `45` hífen, `47` barra. Eles voltam à saída na posição original.
`autoIgnoreCharacters`	Quando verdadeiro, detecta sozinho todos os caracteres não-alfanuméricos e os ignora. Dispensa listar códigos ASCII manualmente — recomendado para CPF, CNPJ e CEP.
`allowShortSegments`	Quando verdadeiro, aceita input menor que a soma dos segmentos, processando o que houver. O padrão (falso) lança erro.
`processPreserveBeforeIgnore`	Ordem de operação entre remover ignorados e calcular segmentos `PRESERVE`. Ative quando um separador cai dentro de um segmento preservado e as posições estão saindo deslocadas.

4.4 Regex Decompose

`algorithm.plugin.decompose.RegexDecompose`

O algoritmo mais flexível para texto estruturado. Você define padrões regex; o primeiro que casar com a string **inteira** é aplicado, e cada grupo de captura (parênteses) recebe uma ação independente — preservar, apagar, redigir ou passar por outro algoritmo. O texto *entre* os grupos é preservado automaticamente.

FORMATO DE ENTRADA

> Qualquer string. Os padrões são testados na ordem declarada, até um casar. Não casando nenhum, aplica-se o `fallbackAction`.

ENTRADA → SAÍDA

ENTRADA · REGEX		SAÍDA
`usuario@empresa.com.br` `([^@]+)(@[^@]+) → REDACT, PRESERVE`	→	`***@empresa.com.br`
`000001999191111` `(\d{6})(\d{9}) → PRESERVE, REDACT "X"`	→	`000001XXXXXXXXX`
`"000001999191111  "` `mesmo padrão, trimInput: true`	→	`"000001XXXXXXXXX  "`
`123456789` `(\d{3})(\d+) → PRESERVE, TRUNCATE`	→	`123`
`ABC` `sem fallbackAction`	→	`Erro: No pattern matched and no fallbackAction is defined`
`ABC` `fallback REDACT "[INVALIDO]"`	→	`[INVALIDO]`

PARÂMETROS

`maskPatterns`	Lista de padrões testados **em ordem**. O primeiro cujo regex casar com a string inteira é o usado — os demais são ignorados. É esta ordenação que permite roteamento condicional (ver nota abaixo).
↳ **`regex`**	Expressão regular que precisa casar com a string **inteira** (ancorada implicitamente). Os grupos de captura delimitam as partes que recebem ações.
↳ **`actions`**	Uma ação por grupo de captura, na mesma ordem dos parênteses. Dois grupos exigem exatamente duas ações.
↳ **`actions.type`**	**PRESERVE** — mantém o trecho. **TRUNCATE** — remove o trecho (vira string vazia). **REDACT** — substitui por texto ou caractere fixo. **APPLY_ALGORITHM** — passa o trecho por outro algoritmo.
↳ **`actions.redactString`**	Texto fixo que substitui o grupo *inteiro*, independentemente do tamanho original. Ex.: `"***"`. Só com `REDACT`.
↳ **`actions.redactCharacter`**	Caractere único que substitui *cada* caractere do grupo, preservando o comprimento. Ex.: `"X"` transforma 9 dígitos em `XXXXXXXXX`. Mutuamente exclusivo com `redactString`.
↳ **`actions.algorithm`**	Instância de algoritmo aplicada ao grupo, quando o tipo é `APPLY_ALGORITHM`. Aceita nomes do catálogo built-in (apêndice B) ou de testes salvos no Tester.
`fallbackAction`	Ação aplicada ao valor inteiro quando nenhum padrão casa. Tem os mesmos quatro tipos, com `redactString`, `redactCharacter` e `algorithm` próprios.

`trimInput`	Remove espaços das extremidades *antes* de tentar casar. O padding é restaurado na saída (ver terceira linha da tabela). Essencial para colunas `CHAR(n)`, que vêm preenchidas com espaços à direita.
`requireMask`	Quando ativo, lança erro se nenhum padrão casar, em vez de usar o fallback. Use quando valores fora do formato devem ser rejeitados em vez de passarem adiante.
`maxInputLength`	Comprimento máximo aceito na entrada. Valores maiores viram erro. Vazio não impõe limite.

Dois erros comuns.
① `NullPointerException … action.type is null` — ao adicionar uma ação pelo formulário, o campo **Tipo de ação** começa vazio; selecione-o explicitamente.
② `Fallback action may not be PRESERVE when requireMask is set` — `requireMask` tem padrão **true**; para usar `fallbackAction: PRESERVE`, defina `requireMask: false` explicitamente.

Roteamento condicional por conteúdo. Como os padrões são testados em ordem, dá para desviar o valor para algoritmos diferentes conforme o que ele contém. Para aplicar um algoritmo quando há `@` e outro quando não há:

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

O primeiro padrão só casa com valores que tenham `@`; todo o resto cai no segundo, que funciona como catch-all.

4.5 String Algorithm Chain

`algorithm.plugin.stringAlgorithmChain.StringAlgorithmChain`

Encadeia algoritmos em sequência: a saída de um vira a entrada do próximo. Exige no mínimo dois. Serve para compor transformações que nenhum algoritmo isolado oferece — mascarar e depois normalizar, ou aplicar duas camadas de substituição.

FORMATO DE ENTRADA

> String compatível com o *primeiro* algoritmo da cadeia. Os seguintes recebem o que o anterior produziu.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`Joao Silva`	→	**`Jeroen`**

Cadeia `FirstName` → `LastName` : o primeiro algoritmo produziu um nome, e o segundo tratou esse resultado como se fosse um sobrenome.

PARÂMETROS

`algorithmReferences`	Lista ordenada de referências a instâncias, **mínimo 2**. Ex.: `[{"name":"dlpx-core:FirstName"}, {"name":"dlpx-core:LastName"}]` . Os nomes vêm do catálogo built-in (apêndice B) ou de testes salvos no Tester.

> **A ordem importa e o encaixe também.** Cada algoritmo precisa aceitar o formato que o anterior produziu. Encadear um algoritmo de data depois de um de nome, por exemplo, falha — o segundo recebe texto que não consegue interpretar.

4.6 Shuffle

`algorithm.plugin.shuffle.Shuffle` ⚠ NÃO DETERMINÍSTICO MODO LOTE

Redistribui os valores **entre** os registros: cada linha recebe o valor que pertencia a outra. Nenhum valor é inventado — o conjunto permanece idêntico, só as associações mudam. Isso preserva perfeitamente a distribuição estatística da coluna, o que o torna ideal para colunas usadas em agregações.

FORMATO DE ENTRADA

> Múltiplos valores de uma vez (modo lote). Cada valor precisa ter comprimento ≥ `minimumShuffleSize`.

ENTRADA → SAÍDA

LOTE DE ENTRADA		LOTE DE SAÍDA
Maria Silva	→	**Pedro Lima**
João Costa	→	**Ana Santos**
Pedro Lima	→	**João Costa**
Ana Santos	→	**Carlos Pereira**
Carlos Pereira	→	**Maria Silva**

PARÂMETROS

`minimumShuffleSize` — Comprimento mínimo para um valor participar do embaralhamento (padrão `3`). Valores mais curtos ficam de fora e passam sem alteração.

> **Não é determinístico — de propósito.** A cada execução a permutação muda. Isso é uma decisão de segurança: uma permutação reproduzível permitiria reexecutar o shuffle e reverter a anonimização. A consequência prática é que o Shuffle **não preserva integridade referencial** entre tabelas nem entre execuções.

PARTE 5

Texto aberto

Para campos de texto livre — observações, laudos, comentários — onde o dado sensível está embutido em meio a texto que precisa continuar legível.

5.1 Free Text Redaction

`algorithm.plugin.freeTextRedaction.FreeTextRedaction`

Varre um texto livre procurando entidades sensíveis — por expressão regular e/ou por uma lista de termos em arquivo — e substitui apenas os trechos encontrados. Todo o resto do texto é preservado, mantendo o documento legível e útil para análise.

FORMATO DE ENTRADA

> Texto livre de qualquer tamanho. Ex.: `Contate João pelo e-mail joao@empresa.com ou CPF 123.456.789-09.`

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`Contate João pelo e-mail joao.silva@empresa.com.`	→	**`Contate João pelo e-mail [REDACTED].`**

PARÂMETROS

`regularExpressions`	Lista de objetos `{patternString}` com as regex que identificam trechos a redigir. Cada padrão é aplicado ao texto inteiro, em todas as ocorrências.
`isDenyList`	**true** — lista de bloqueio: redige o que *casa* com os padrões. **false** — lista de permissão: redige tudo que *não* casa. A segunda opção é mais segura para textos imprevisíveis, mas costuma destruir a legibilidade.
`regExRedactValue`	Texto que substitui os trechos capturados pelas expressões regulares. Ex.: `[REDACTED]`.
`lookupFile`	Arquivo com termos literais, um por linha, buscados no texto. Serve para nomes próprios e termos que regex não descreve bem.
`lookupFileRedactValue`	Texto que substitui as ocorrências vindas do arquivo. É independente do `regExRedactValue`, permitindo marcar visualmente a origem de cada redação.

5.2 Redact

`algorithm.plugin.redact.Redact`

Versão mais simples e direta da redação: um mapa de `regex → texto de substituição`. Cada chave do mapa é um padrão, e o valor correspondente é o que entra no lugar. Sem regex configurada, devolve o valor de entrada sem mascaramento algum.

FORMATO DE ENTRADA

> Qualquer string. Trechos que casarem com as regex são substituídos.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`Contate João pelo e-mail joao.silva@empresa.com.`	→	**`Contate João pelo e-mail [EMAIL].`**

PARÂMETROS

`regexRedact`	Mapa de `{regex → texto_substituição}`. Permite marcadores diferentes por tipo de dado — `[EMAIL]`, `[CPF]`, `[TELEFONE]` — em uma única configuração.

> **Configuração vazia não mascara.** Sem nenhuma regex em `regexRedact`, o algoritmo devolve a entrada intacta e **não** emite aviso. Verifique sempre que o mapa foi preenchido.

PARTE 6

Financeiro

Instrumentos financeiros com estrutura validável: cartões com dígito Luhn e IBANs com checksum de país.

6.1 Payment Card

`algorithm.plugin.characterMapping.PaymentCard` **DETERMINÍSTICO**

Mascara números de cartão preservando o **BIN** (os primeiros dígitos, que identificam bandeira e emissor) e, opcionalmente, os últimos dígitos usados em conferências. O número gerado mantém o **dígito verificador Luhn válido**, então continua passando em validações de formulário e de sistema.

FORMATO DE ENTRADA

> Número de cartão com ou sem espaços/hífens. Ex.: `4111 1111 1111 1111`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
4111111111111111	→	**4111062777755578**
5500000000000004	→	**5500836779524256**
4111111111111111 `(preserve: 0)`	→	**9393500626048800**

Nas duas primeiras o BIN (`4111` , `5500`) foi preservado automaticamente. Na terceira, com `preserve: 0` , até o BIN mudou — a bandeira do cartão deixa de ser identificável.

PARÂMETROS

`preserve`	Quantos dígitos do *final* do cartão preservar. Tipicamente `4` , para manter os últimos quatro que aparecem em extratos e telas de confirmação.
`minMaskedPositions`	Número mínimo de posições que devem efetivamente mudar. Garante que o cartão mascarado não fique parecido demais com o original.

> **Preservar os 4 finais tem custo.** Últimos-4 mais BIN mais data de transação costuma ser suficiente para reidentificar um cartão em bases pequenas. Use `preserve: 4` só quando o processo de negócio realmente depender disso.

6.2 IBAN

`algorithm.plugin.iban.IBAN` **DETERMINÍSTICO**

Mascara um IBAN preservando o código do país e recalculando o dígito verificador, de modo que o resultado continue sendo um IBAN estruturalmente válido. Você controla quantos caracteres da parte nacional são embaralhados.

FORMATO DE ENTRADA

> IBAN no formato padrão, sem espaços. Ex.: `GB29NWBK60161331926819`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`GB29NWBK60161331926819` (mask 20)	→	**`GB59DNLT98906202195739`**
`GB29NWBK60161331926819` (mask 8)	→	**`GB18NWBK60161374219890`**

Com `numCharsToMask: 8` o código do banco (`NWBK`) e a agência sobrevivem; com `20`, praticamente só o país permanece. O `GB` é sempre preservado e o checksum recalculado.

PARÂMETROS

`validateInput` *	Quando verdadeiro, valida o checksum do IBAN antes de mascarar e rejeita entradas inválidas. Desligue apenas se a base tiver IBANs sabidamente malformados que ainda assim precisam ser processados.
`numCharsToMask` *	Quantos caracteres embaralhar, contados a partir do fim da parte nacional. Valores baixos preservam banco e agência; valores altos mascaram tudo depois do país.

PARTE 7

Lookup & Mapeamento

Algoritmos que substituem valores consultando uma fonte externa — um arquivo de substitutos, uma tabela de-para, ou um banco de mapeamentos persistente.

7.1 Secure Lookup

`algorithm.plugin.secureLookup.SecureLookup` **DETERMINÍSTICO** **⚠ EXCETO COM RANDOMIZE** **REQUER ARQUIVO**

Substitui o valor por uma linha de um arquivo de lookup, escolhida pelo hash de (valor + chave). O mesmo input sempre seleciona a mesma linha, o que dá consistência referencial entre tabelas sem precisar de banco de mapeamento. É o algoritmo de lookup mais usado do plugin.

FORMATO DE ENTRADA

> Qualquer string não nula. O conteúdo do arquivo determina o domínio da saída.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`João da Silva`	→	**`Pedro`**
`Maria Souza`	→	**`Rafael`**
`João da Silva` (ALL_UPPER)	→	**`PEDRO`**

A terceira linha mostra que `maskedValueCase` só afeta a capitalização — a linha selecionada do arquivo continua a mesma.

PARÂMETROS

`lookupFile` *	Arquivo com um valor substituto por linha. O número de linhas define a diversidade da saída: um arquivo com 20 nomes fará muitos valores distintos colidirem no mesmo substituto.
`hashMethod`	Como derivar o índice da linha. **SHA256** — hash criptográfico, determinístico pela chave (recomendado). **LEGACY** — método antigo, mantido para compatibilidade com dados já mascarados. **RANDOMIZE** — seleção aleatória, *não* determinística: cada execução muda o resultado.
`maskedValueCase`	Capitalização da saída. **PRESERVE_LOOKUP_FILE** — como no arquivo. **PRESERVE_INPUT** — copia o padrão do input. **ALL_LOWER** / **ALL_UPPER**.
`inputCaseSensitive`	Quando verdadeiro, "João" e "JOÃO" produzem hashes distintos e podem receber substitutos diferentes. Deixe falso para tratar variações de caixa como o mesmo valor.
`trimWhitespaceFromInput`	Remove espaços das extremidades do input antes do hash. Faz `"  João  "` e `"João"` caírem no mesmo substituto — importante em colunas `CHAR(n)`.
`trimWhitespaceInLookupFile`	Remove espaços das extremidades de cada linha do arquivo ao carregá-lo, evitando que linhas com espaço sobrando virem valores distintos.

7.2 Null-Safe Secure Lookup

`algorithm.plugin.nullSecureLookup.NullSecureLookup` **LIMITADO NO RUNNER**

Variante do Secure Lookup com tratamento explícito de valores nulos: input nulo devolve nulo, em vez de erro. Não expõe parâmetros — usa o arquivo de lookup embutido no plugin.

FORMATO DE ENTRADA

> Qualquer string, ou nulo.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`valor-sensivel`	→	`null` (no runner standalone)

PARÂMETROS

Nenhum.

> **No runner standalone sempre devolve `null`.** O algoritmo depende do contexto do Masking Engine para resolver o arquivo de lookup embutido. O resultado `null` aqui é esperado e não indica erro de configuração.

7.3 Data Cleansing

`algorithm.plugin.dataCleansing.DataCleansing` **REQUER ARQUIVO**

Substitui valores segundo uma tabela de-para **explícita**: você declara exatamente qual valor vira qual. Ao contrário do Secure Lookup, não há hash nem sorteio. Na prática é mais uma ferramenta de padronização que de mascaramento — normalizar siglas, unificar grafias divergentes.

FORMATO DE ENTRADA

> String com correspondência no arquivo. Sem correspondência, o valor original passa **sem alteração**.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
SP	→	**São Paulo**
rj	→	**Rio de Janeiro**
XX	→	**XX** (sem correspondência)

A segunda linha casou apesar da caixa diferente porque `caseSensitive` está falso.

PARÂMETROS

lookupFile *	Arquivo com um mapeamento por linha, no formato `original{delimitador}substituto`. Ex.: `SP,São Paulo`.
delimiter	Separador entre original e substituto. Ex.: `,` para CSV, `;`, ou tabulação. Omitido, usa tabulação.
caseSensitive	Quando verdadeiro, `SP` e `sp` exigem entradas separadas no arquivo. Falso é o mais prático na maioria dos casos.
trimWhitespace	Remove espaços das extremidades do input e das linhas do arquivo antes de comparar, evitando falhas de correspondência por padding do banco.

> **Valores sem correspondência passam intactos.** Isso significa que uma tabela de-para incompleta deixa dados originais vazarem **silenciosamente**. Se a coluna é sensível, valide a cobertura do arquivo antes de rodar em produção.

7.4 Mapping

`algorithm.plugin.mapping.Mapping` **NÃO FUNCIONA NO RUNNER**

Mantém um mapeamento persistente 1-para-1 em banco de dados: cada valor original recebe um substituto único, gravado e reutilizado em todas as tabelas e execuções futuras. É a garantia mais forte de **consistência referencial** do plugin — e a única que sobrevive entre jobs distintos.

FORMATO DE ENTRADA

> Qualquer string. O algoritmo consulta o mapping set e, se o valor for novo, cria e persiste um substituto.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`cliente@email.com`	→	`Erro: Mapping set references are not supported`

PARÂMETROS

`mappingSet` *	Referência ao conjunto de mapeamento que armazena as correspondências.
↳ **`algorithmName`** *	Nome do mapping set no Masking Engine — identifica qual conjunto usar. Ex.: `client-name-mapping`.
↳ **`host`**	Host do banco onde o mapping set está armazenado, quando remoto.
↳ **`port`**	Porta do banco do mapping set remoto.
↳ **`database`**	Nome do banco de dados remoto.
↳ **`schema`**	Schema do banco remoto.
↳ **`isRemote`**	Quando verdadeiro, usa os dados de conexão acima. Falso usa a instância local do Engine.
↳ **`mappingLookupKey`**	Chave de partição que permite ao mesmo mapping set servir múltiplos contextos isolados — por cliente, por ambiente.
↳ **`propertiesRef`**	Arquivo de propriedades com a configuração de conexão, alternativa a preencher host/porta/banco/schema individualmente.
`ignoreCharacters`	Códigos ASCII a remover do input antes da consulta. Ex.: `45` hífen, `46` ponto — faz CPF com e sem formatação apontar para o mesmo mapeamento.

> **Indisponível no Algorithm Tester.** O runner standalone não tem banco de mapeamento; qualquer teste devolve `Mapping set references are not supported`. Este algoritmo só pode ser validado no Masking Engine.

PARTE 8

Multi-Column

Algoritmos que recebem a linha inteira, não um valor isolado — permitindo que o mascaramento de uma coluna dependa do conteúdo de outra.

8.1 Multi-Column Condition

`algorithm.plugin.conditional.MultiColumnCondition` **DETERMINÍSTICO** **MULTI-COLUNA**

Escolhe qual algoritmo aplicar a cada coluna **conforme o valor de uma coluna condicional**. O caso clássico: se a coluna de gênero disser "F", mascarar o nome com uma lista de nomes femininos; se disser "M", com nomes masculinos — mantendo a coerência interna do registro.

FORMATO DE ENTRADA

> Uma linha com colunas nomeadas. Obrigatória: `key` (a coluna condicional). Opcionais: `string1` … `string10`, `numeric1` … `numeric3`, `date1` … `date3`, `binary1` … `binary3`.

ENTRADA → SAÍDA

LINHA DE ENTRADA		LINHA DE SAÍDA
`key = "F"` `string1 = "Maria"`	→	**`key = "F"`** **`string1 = "Jalisa"`**

A coluna `key` não é mascarada — ela só decide qual condição se aplica. Apenas `string1` tinha algoritmo configurado.

PARÂMETROS

`conditions`	Lista de condições avaliadas contra o valor de `key`. Cada condição declara os valores que a ativam e quais algoritmos aplicar a quais slots de coluna.
↳ **`key`**	Lista de valores da coluna condicional que ativam esta condição. Ex.: `["F", "Female"]` ativa para qualquer um dos dois.
↳ **`string1…string10`**	Algoritmo aplicado à coluna `stringN` quando a condição é ativada. Ex.: `{"name": "dlpx-core:FirstName"}`. Slots sem algoritmo passam intactos.
↳ **`numeric1…numeric3`**	Algoritmo aplicado às colunas numéricas (`BigDecimal`).
↳ **`date1…date3`**	Algoritmo aplicado às colunas de data (`LocalDateTime`).
↳ **`binary1…binary3`**	Algoritmo aplicado às colunas binárias (`ByteBuffer`).
`fallbackAlgo`	Algoritmo aplicado às colunas string quando *nenhuma* condição casa. Sem ele, os valores passam sem mascaramento.
`fallbackKey`	Valor assumido como chave quando a coluna `key` está nula ou ausente na linha.
`keyCaseSensitive`	Quando verdadeiro, a comparação de `key` com as listas das condições diferencia maiúsculas de minúsculas. Padrão: falso.
`filterLength`	Usa apenas os primeiros (ou últimos) *N* caracteres de `key` na comparação. Útil quando a coluna condicional tem prefixo significativo e sufixo variável.
`filterDirection`	De onde contar os caracteres de `filterLength`: `LEFT` (do início) ou `RIGHT` (do fim).

> **De onde vêm os nomes `key`, `string1` …** São **slots fixos** do algoritmo, não nomes de colunas do seu banco. A tradução entre a coluna real (`GENERO`, `NOME`) e o slot (`key`, `string1`) é feita no **inventário** do Masking Engine. No Algorithm Tester você simula esse mapeamento preenchendo a tabela de entrada manualmente.

> **Colunas sem algoritmo não são mascaradas.** Se uma condição declara apenas `string1`, todas as demais colunas da linha passam intactas — mesmo contendo dado sensível. Configure `fallbackAlgo` ou declare cada slot explicitamente.

8.2 Multi-Column Address

`algorithm.plugin.address.MultiColumnAddress` **REQUER ARQUIVO DE ENDEREÇOS**

Mascara campos de endereço espalhados por várias colunas — logradouro, número, bairro, cidade, estado, CEP — gerando um endereço sintético **coerente entre si**. Sem ele, mascarar cada coluna isoladamente produziria combinações impossíveis (rua de São Paulo com CEP do Amazonas).

FORMATO DE ENTRADA

> Colunas de endereço da linha. Exige um arquivo de lookup de endereços no formato específico do Delphix.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`Rua das Flores, 123`	→	`Erro: FileReference nulo — arquivo de endereços não configurado`

PARÂMETROS

Não documentados neste guia — o algoritmo não inicializa sem o arquivo de endereços, o que impede inspecionar seu schema no runner.

> **Não testável no runner standalone.** O algoritmo falha na inicialização sem o arquivo de lookup de endereços, que não acompanha o plugin. Configure-o e valide diretamente no Masking Engine.

PARTE 9

Outros

Dois algoritmos especializados: recálculo genérico de dígito verificador e tokenização reversível.

9.1 Check Digit

`algorithm.plugin.checkdigit.Checkdigit` **DETERMINÍSTICO**

Mascara números que carregam dígito verificador e **recalcula o verificador** depois, para que o resultado continue passando na validação. É genérico: qualquer esquema de soma ponderada + módulo pode ser descrito por seus parâmetros — código de barras, EAN, boleto, matrícula, inscrição estadual.

FORMATO DE ENTRADA

> String numérica sem pontuação, incluindo o dígito verificador na posição configurada. Ex.: `7891000315507` (EAN-13).

ENTRADA → SAÍDA

ENTRADA		SAÍDA
123456789012	→	**299460400954**

PARÂMETROS

`weightList` *	Pesos aplicados a cada dígito na soma ponderada, um por dígito de dados. Ex.: EAN-13 usa `[1,3,1,3,1,3,1,3,1,3,1,3]`. O tamanho da lista deve ser igual a `numDigitsForCheckdigitCalculation`.
`modulusNumber` *	Divisor do módulo. O verificador é normalmente `(módulo − resto) % módulo`. EAN-13 usa `10`; CNPJ usa `11`.
`checkDigitIndex` *	Posição do dígito verificador na string, contada a partir de `0` à esquerda. Num código de 13 dígitos com verificador no fim, informe `12`.
`calculateChecksumRightToLeft` *	Direção em que os pesos são aplicados. **true** — direita para esquerda (boletos, CNPJ). **false** — esquerda para direita (EAN, UPC).
`numDigitsForCheckdigitCalculation` *	Quantos dígitos entram na soma, excluindo o próprio verificador. EAN-13 tem 12 dígitos de dados.
`swapModulusForZeroRemainder`	Quando o resto é zero, usa o próprio `modulusNumber` como verificador em vez de `0`. Alguns padrões (CNPJ entre eles) adotam essa regra.
`checksumCalculationType`	**STANDARD** — soma ponderada módulo N (padrão). **TFN** — variante australiana (Tax File Number), com lógica própria.
`numericAlgorithm`	Algoritmo que embaralha os dígitos que não são o verificador. Omitido, usa o Character Mapping numérico embutido.
`alphaNumericAlgorithm`	Algoritmo usado quando a entrada contém letras além de dígitos. Sem ele, entradas alfanuméricas podem falhar conforme o `characterHandling`.
`fallbackAlgorithm`	Algoritmo acionado com entrada inválida quando `invalidInputHandling = FALLBACK_MASK`. Recebe o valor original inteiro.
`preserveRegex`	Regex que identifica partes a preservar — prefixos fixos, códigos de país. Os trechos que casam ficam intactos e só o restante é mascarado.
`inputHandlingConfig`	Bloco com o tratamento da entrada antes do mascaramento.
↳ **`characterHandling`**	**NUMERIC_ONLY** — aceita só `0–9`. **STANDARD** — letras usam seu valor ASCII. **ASCII_VALUE_MINUS_48** — letras usam `(ASCII − 48)`, usado por padrões que intercalam letras e dígitos.

↳ **`invalidInputHandling`**	**ERROR** — lança exceção (padrão). **FALLBACK_MASK** — delega ao `fallbackAlgorithm`.
↳ **`shortInputHandling`**	Entrada mais curta que o esperado: **FALLBACK** delega, **PAD_LEFT** / **PAD_RIGHT** completam com `padCharacter`.
↳ **`padCharacter`**	Caractere de preenchimento das entradas curtas. Normalmente `0`.
↳ **`trimWhitespace`**	Remove espaços das extremidades antes de processar. Necessário para colunas `CHAR(n)` com padding.

9.2 Tokenization

`algorithm.plugin.tokenization.Tokenization` **REVERSÍVEL** **⚠ NÃO DETERMINÍSTICO POR PADRÃO**

O único algoritmo **reversível** do conjunto. Cifra o valor com AES e devolve um token; com a mesma chave e configuração, o valor original pode ser recuperado (no Tester, pelo botão *Detokenizar* — modo `REIDENTIFY`). É a escolha quando o dado precisa voltar ao original em algum ponto do fluxo.

FORMATO DE ENTRADA

> String alfanumérica. Caracteres fora do conjunto tokenizável — símbolos, espaços, hífens — acionam o `fallback`.

ENTRADA → SAÍDA

ENTRADA		SAÍDA
`4111111111111111`	→	`AN0xIV5riitfgy/WeQAd4M7pZgrIDoxO`
`AN0xIV5riitfgy/WeQAd4M7pZgrIDoxO` (REIDENTIFY)	→	`4111111111111111`
`4111-1111-1111-1111` (com hífens)	→	`r/KCnIICoKdHrWGIThHCpypp77gZgLBtOfZB`

A segunda linha demonstra a reversão exata. Note que o token é **mais longo** que o original — ele carrega o vetor de inicialização.

PARÂMETROS

`fallback`	O que fazer com caracteres não tokenizáveis. **NONE** — lança erro. **CHARACTER_MAPPING** — aplica mascaramento genérico nesses caracteres e segue.
`ivLength`	Tamanho do vetor de inicialização AES em bytes (padrão `8`). **É este parâmetro que decide se o algoritmo é determinístico.** Com qualquer valor maior que zero, um IV aleatório é sorteado a cada execução: a mesma entrada com a mesma chave gera um token diferente toda vez. Com `0`, não há IV e o token passa a ser sempre o mesmo. Valores maiores aumentam a aleatoriedade, mas consomem mais espaço no resultado.
`cmCharacterGroups`	Só com `fallback = CHARACTER_MAPPING`. Define quais caracteres são intercambiáveis no fallback. Vazio usa os grupos padrão.
`cmMinMaskedPositions`	Só com `fallback = CHARACTER_MAPPING`. Mínimo de posições que o fallback deve mascarar, evitando tokens em que quase nada mudou.

> ⚠ **Não determinístico com a configuração padrão.** Com `ivLength` maior que zero — e o padrão é `8` — cada execução sorteia um vetor de inicialização novo, então a mesma entrada com a mesma chave produz um token diferente toda vez. Todos revertem corretamente, mas **não servem para preservar joins**: o mesmo valor em duas tabelas vira dois tokens distintos. Se precisar que o mesmo valor sempre gere o mesmo token, use `ivLength: 0` — aí a saída é determinística e continua reversível.

> **O token não preserva o formato.** Apesar de ser descrito como *format-preserving*, o resultado observado é uma string Base64 mais longa que a entrada — `4111111111111111` (16 caracteres) virou 32. Dimensione a coluna de destino com folga, ou o job falhará por truncamento.

> **A chave é o segredo.** Quem tiver a chave reverte qualquer token. Trate-a com o mesmo rigor de uma chave de produção: fora do código, fora do repositório, com rotação controlada — e lembre que rotacionar invalida todos os tokens já emitidos.

APÊNDICE A

Conceitos transversais

Comportamentos que atravessam vários algoritmos e costumam causar confusão quando encontrados pela primeira vez.

DETERMINISMO E O PAPEL DA CHAVE

A maioria dos algoritmos é **determinística**: o mesmo valor de entrada, com a mesma chave, sempre produz a mesma saída. É isso que preserva junções — se `João` vira `Ana` na tabela de clientes, vira `Ana` também na de pedidos.

A chave é o que torna o resultado imprevisível para quem não a possui. Trocá-la muda toda a saída, como visto em §4.1. As duas consequências práticas: **a mesma chave precisa ser usada em todo o projeto**, e **trocar a chave invalida a consistência com dados já mascarados**.

COMPORTAMENTO	ALGORITMOS
Determinístico por chave	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
⚠ **Não determinístico**	**Shuffle** (§4.6) — sempre; permutação diferente a cada execução. **Tokenization** (§9.2) — com `ivLength` maior que zero, e o padrão é `8`; use `ivLength: 0` para saída determinística. **Secure Lookup** (§7.1) — apenas com `hashMethod: RANDOMIZE`.
Independe de chave	Min/Max (ambos), Redact, Free Text Redaction, Data Cleansing, Repeat First Digit, Numeric Expression sem `seed`

ALGORITMOS QUE PODEM NÃO MASCARAR NADA

Vários algoritmos devolvem o valor original silenciosamente em certas condições. Nenhum deles emite aviso — a verificação é sua.

ALGORITMO	QUANDO O DADO PASSA INTACTO
Min/Max BigDecimal · Date/Time	Sempre que o valor já está dentro da faixa
Data Cleansing	Valor sem correspondência no arquivo de-para
Redact	`regexRedact` vazio, ou nenhum padrão casando
Regex Decompose	`fallbackAction: PRESERVE` com valor que não casa nenhum padrão
Full Name	`firstNameAlgorithmRef` ou `lastNameAlgorithmRef` não configurado
Multi-Column Condition	Slot de coluna sem algoritmo, ou nenhuma condição casando sem `fallbackAlgo`
Character Mapping	Caracteres fora de todos os `characterGroups` (acentos, pontuação)

COLUNAS CHAR(N) E O PADDING INVISÍVEL

Colunas `CHAR(n)` preenchem o valor com espaços à direita até o tamanho declarado. Um CPF de 11 dígitos numa coluna `CHAR(15)` chega ao algoritmo como `"12345678909    "`. Como `\d` não casa com espaço, regex que funcionam no Tester falham em produção com mensagens como `No pattern matched` ou `did not conform to the expected format`.

A correção é ativar a opção de trim do algoritmo: `trimInput` no Regex Decompose, `trimWhitespace` no Check Digit e no Data Cleansing, `trimWhitespaceFromInput` no Secure Lookup. O padding é

restaurado na saída.

REFERÊNCIAS ENTRE ALGORITMOS

Muitos parâmetros pedem uma "referência a algoritmo" — `firstNameAlgorithmRef`, `actions.algorithm`, `numericAlgorithm`, `fallbackAlgo`. O valor esperado é `{"name": "<instância>"}`, onde a instância pode ser:

① um nome do catálogo built-in, prefixado por `dlpx-core:` (apêndice B); ② o nome de um teste salvo no próprio Algorithm Tester; ③ no Masking Engine, qualquer instância de algoritmo configurada. Nomes inexistentes produzem `Sub-algorithm not found`.

APÊNDICE B

Catálogo de instâncias built-in

As 62 instâncias que acompanham o plugin, utilizáveis em qualquer parâmetro de referência. Prefixe com `dlpx-core:` — por exemplo `dlpx-core:FirstName`. Nomes com espaço funcionam normalmente.

- 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

> **Sufixo `SL`** indica *Secure Lookup* — instâncias pré-configuradas com arquivos de lookup temáticos embutidos (profissões, países, estado civil). São a forma mais rápida de mascarar colunas categóricas sem montar arquivo próprio.

APÊNDICE C

Limitações do runner standalone

O Algorithm Tester executa os algoritmos fora do Masking Engine. Alguns recursos dependem do contexto do Engine e não funcionam localmente.

RECURSO	SITUAÇÃO
`FileReference`	**Suportado.** Use URI `file:///caminho/absoluto`. Requer `commons-codec` em `lib/`. Os arquivos são gerenciados em *Configurações → Arquivos*.
`MappingSetReference`	**Não suportado.** O algoritmo *Mapping* (§7.4) falha com `Mapping set references are not supported` — depende de banco de mapeamento.
`EmbeddedGeneratorReference`	**Não suportado.**
`MaskValueMetadata`	Retorna sempre `null`.
Multi-Column Address (§8.2)	Não inicializa sem o arquivo de lookup de endereços do Delphix.
Null-Safe Secure Lookup (§7.2)	Devolve sempre `null` — depende do arquivo embutido resolvido pelo Engine.

> **Sobre as saídas deste guia.** Todos os pares entrada → saída foram gerados executando os algoritmos no `AlgorithmRunner` com o plugin `delphix-algorithm-plugin 2026.3.0-SNAPSHOT`. Algoritmos determinísticos reproduzem exatamente estes valores com a mesma chave; o Shuffle, por natureza, produzirá uma permutação diferente a cada execução.