Introduction to Transformers

Transformers are **crucial components** within your data ingestion pipeline, responsible for **standardizing and refining raw data** fetched by the crawlers. They act as an intermediary layer between data extraction and data destination.

They are used in the **Scheduler SDK's** **IngestionOrchestrator** (orchestrator.ts). After a DataSource (crawler) fetches raw data, this raw data is passed to a configured transformer function. The transformer processes this data and returns it in a consistent IngestionData format, which is then sent to the destination plugin.

Standard Transformer Structure

All transformer functions must adhere to the following **standard TypeScript signature**:

```
TypeScript

export type IngestionDataTransformer = (rawData: any[], initialPayload?: any) =>
Promise<IngestionData[]>;
```

- **rawData: any[]**: This is an array of raw data items returned directly by a crawler's execute method. The type any[] is used for flexibility as raw data can vary greatly between sources (e.g., raw file content, API responses, parsed HTML).
- **initialPayload?: any**: This optional object contains the original payload that triggered the task (e.g., webhook data, cron event context). It can be useful if the transformer needs additional context beyond the raw data itself (e.g., changeType from a webhook).
- **Promise<IngestionData[]>**: The transformer must always return a Promise that resolves to an array of IngestionData objects. This ensures a consistent output format for the next stage of the pipeline.

Creating Custom Transformer Functions

Users can easily **create their own custom transformer functions** by adhering to this standard IngestionDataTransformer signature. This allows for complete flexibility in how raw data is processed and shaped to meet specific requirements.

To implement a custom transformer:

1. Define an async function that accepts rawData and initialPayload as parameters.
2. Inside the function, apply your desired logic to transform rawData.
3. Return a Promise that resolves to an array of IngestionData objects.
4. Register this custom transformer with the Scheduler SDK using globalIngestionManager.registerSource() (in hybrid mode) or by adding it to the _defaultCrawlerRegistry for automated registration.

This modular design ensures that the data transformation layer is highly extensible and adaptable to diverse data processing needs.

Default Transformers:

gitcodeMetadataExtractorTransformer

This transformer is designed to enrich IngestionData items originating from code repositories. It intelligently **identifies the programming language** of a file by examining its file extension, leveraging a predefined mapping. Furthermore, for recognized code or text files with actual content, it generates a **short code preview** by extracting the first few lines. Both the identified language and the codePreview are then appended to the item's metadata attribute. Files that are not identified as code or text (e.g., binary files) or have no content are simply passed through the transformation pipeline without modification.

gdriveContentNormalizerTransformer

This transformer is responsible for **standardizing content from Google Drive items** into a more usable plain text format. It intelligently handles various content types: for **PDF files**, it extracts text using the pdf-parse library; for other binary content, it attempts to convert it to a UTF-8 string. If the content is an object, it's converted to a JSON string. The transformer also **updates the mimeType metadata** to reflect the new content type and adds a transformedBy flag. Items that cannot be processed are passed through with an error flag in their metadata, ensuring graceful failure.

genericIngestionPreprocessor

This transformer serves as a **universal preprocessing step** for all IngestionData items in the pipeline. It ensures data consistency by **converting all content to a UTF-8 string** (handling Buffer objects gracefully). It then **normalizes whitespace** within the text content and calculates the contentLength. Finally, it enriches the metadata of each item by adding an

ingestionTimestamp and the calculated contentLength, along with a flag indicating it has been processed by this generic preprocessor.

htmlToPlaintextTransformer

This transformer is designed to **convert HTML content into clean, readable plain text**. It intelligently identifies HTML content based on MIME types or structural heuristics (e.g., starting with <html). For identified HTML, it first **removes all <script> and <style> tags** to eliminate non-textual elements. It then strategically **adds newlines after common block-level HTML elements** (like paragraphs, headings, list items) to preserve readability and structure in the plain text output. Finally, it extracts the text content from the <body> and **normalizes all whitespace**, ensuring a clean and consistent plain text representation. Items that are not HTML or have no content are passed through the pipeline unchanged.

passthroughTransformer

This transformer is the **simplest form of data transformation**, designed to **return IngestionData items exactly as they were received**. It's particularly useful when the data source already provides data in the desired IngestionData format, requiring no further processing before being sent to its destination. Although it performs a .map() operation, this is primarily to create a new array reference, ensuring the original input array isn't directly mutated, while the individual data items remain unchanged.

teamsMessageCleanerTransformer

This transformer specializes in **converting raw Microsoft Teams message content, typically received as HTML, into clean plain text**. It first performs a heuristic check to confirm if the content is indeed HTML. If confirmed, it uses cheerio to parse the HTML, effectively **stripping out all <script> and <style> tags** to remove extraneous code. To maintain readability, it then strategically **inserts newlines after common block-level HTML elements** (like paragraphs

and headings) before extracting the pure text content. Finally, it **normalizes all whitespace** within the extracted text, ensuring a clean and consistent output for further processing. Items that are not HTML or are empty are passed through unchanged, with errors logged for failed transformations.