Amazon Textract

Developer Guide

Amazon Textract: Developer Guide

Table of Contents

What is Amazon Textract?

Amazon Textract allows you to add document text detection and analysis to your applications through simple calls to API endpoints. The Amazon Textract Text Detection API can detect typed and handwritten text in a variety of documents including financial reports, medical records, invoices and receipts, and tax forms. For documents with structured data, you can use the Amazon Textract Document Analysis API to extract text, forms and tables.

Amazon Textract is based on the same proven, highly scalable, deep-learning technology that was developed by Amazon's computer vision scientists to analyze billions of images and videos daily. You don't need any machine learning expertise to use it since Amazon Textract includes simple APIs that can analyze image files and PDF files. Amazon Textract is always learning from new data, and Amazon is continually adding new features to the service.

The following are common use cases for using Amazon Textract:

- **Creating an intelligent search index** – Using Amazon Textract you can create libraries of text that is detected in image and PDF files.
- **Using intelligent text extraction for natural language processing (NLP)** – Amazon Textract enables you to extract text into words and lines. It also groups text by table cells if Amazon Textract document table analysis is enabled. Amazon Textract provides you with control over how text is grouped as an input for NLP applications. It can extract text as words and lines. It also groups text by table cells if Amazon Textract document table analysis is enabled.
- **Accelerating the capture and normalization of data from different sources** – Amazon Textract enables text and tabular data extraction from a wide variety of documents, such as financial documents, research reports, and medical notes. With Amazon Textract Analyze Document APIs, you can easily and quickly extract unstructured and structured data from your documents.
- **Automating data capture from forms** – Amazon Textract enables structured data to be extracted from forms. With Amazon Textract Analysis APIs, you can build extraction capabilities into existing business workflows so that user data submitted through forms can be extracted into a usable format.

Some of the benefits of using Amazon Textract include:

- **Integration of document text detection into your apps** – Amazon Textract removes the complexity of building text detection capabilities into your applications by making powerful and accurate analysis available with a simple API. You don’t need computer vision or deep learning expertise to use Amazon Textract to detect document text. With Amazon Textract Text APIs, you can easily build text detection into any web, mobile, or connected device application.
- **Scalable document analysis** – Amazon Textract enables you to analyze and extract data quickly from millions of documents, which can accelerate decision making.
- **Low cost** – With Amazon Textract, you only pay for the documents you analyze. There are no minimum fees or upfront commitments. You can get started for free, and save more as you grow with our tiered pricing model.

With synchronous processing, Amazon Textract can analyze single-page documents for applications where latency is critical. Amazon Textract also provides asynchronous operations to extend support to multipage documents.

First-Time Amazon Textract Users

If this is your first time using Amazon Textract, we recommend that you read the following sections in order:

1. **How Amazon Textract Works (p. 3)** – This section introduces the Amazon Textract components and how they work together for an end-to-end experience.
2. **Getting Started with Amazon Textract (p. 28)** – In this section, you set up your account and test the Amazon Textract API.

How Amazon Textract Works

Amazon Textract enables you to detect and analyze text in single or multipage input documents (see Input Documents (p. 7)).

Amazon Textract provides operations for detecting text to find data relationships within forms and tables, and for processing invoices and receipts. For more information, see Detecting Text (p. 3), Analyzing Documents (p. 4), and Analyzing Invoices and Receipts (p. 5).

Amazon Textract users with both synchronous operations and asynchronous operations:

- Synchronous operations are for processing small, single-page, documents and with near real-time responses. For more information, see Processing Documents with Synchronous Operations (p. 32).
- Asynchronous operations that you can use to process larger, multipage documents. Asynchronous responses aren't in real time. For more information, see Processing Documents with Asynchronous Operations (p. 108).

When an Amazon Textract operation processes a document, the results are returned in an array of the section called “ Block ” (p. 231) objects or an array of the section called “ ExpenseDocument ” (p. 240) objects. Both objects contain information that's detected about items, including their location on the document and their relationship to other items on the document. For more information, see Amazon Textract Response Objects (p. 8). For examples that show how to use `Block` objects, see Examples (p. 139).

Topics

Detecting Text

Amazon Textract provides synchronous and asynchronous operations that return only the text detected in a document. For both sets of operations, the following information is returned in multiple the section called “ Block ” (p. 231) objects.

- The lines and words of detected text
- The relationships between the lines and words of detected text
- The page that the detected text appears on
- The location of the lines and words of text on the document page

For more information, see the section called “Lines and Words of Text” (p. 11).

To detect text synchronously, use the DetectDocumentText (p. 197) API operation, and pass a document file as input. The entire set of results is returned by the operation. For more information and an example, see Processing Documents with Synchronous Operations (p. 32).

Note
The Amazon Rekognition API operation `DetectText` is different from `DetectDocumentText`. You use `DetectText` to detect text in live scenes, such as posters or road signs.

To detect text asynchronously, use StartDocumentTextDetection (p. 222) to start processing an input document file. To get the results, call GetDocumentTextDetection (p. 206). The results are returned in one or more responses from `GetDocumentTextDetection`. For more information and an example, see Processing Documents with Asynchronous Operations (p. 108).

Analyzing Documents

Amazon Textract analyzes documents and forms for relationships among detected text. Amazon Textract analysis operations return 3 categories of document extraction — text, forms, and tables. The analysis of invoices and receipts is handled through a different process, for more information see Analyzing Invoices and Receipts (p. 5).

Text Extraction

The raw text extracted from a document. For more information, see Lines and words of text (p. 11).

Form Extraction

Form data is linked to text items extracted from a document. Amazon Textract represents form data as key-value pairs. In the following example, one of the lines of text detected by Amazon Textract is *Name: Jane Doe*. Amazon Textract also identifies a key (*Name:*) and a value (*Jane Doe*). For more information, see Form data (Key-value pairs) (p. 13).

Name: Jane Doe

Address: 123 Any Street, Anytown, USA

Birth date: 12-26-1980

Key-value pairs are also used to represent check boxes or option buttons (radio buttons) that are extracted from forms.

Male: ☑

For more information, see Selection elements (p. 17).

Table Extraction

Amazon Textract can extract tables, table cells, and the items within table cells and may be programmed to return the results in a JSON, .csv, or a .txt file.

Name	Address
Ana Carolina	123 Any Town

For more information, see Tables (p. 15). Selection elements can also be extracted from tables. For more information, see Selection elements (p. 17).

For analyzed items, Amazon Textract returns the following in multiple the section called “ Block ” (p. 231) objects:

- The lines and words of detected text
- The content of detected items
- The relationship between detected items
- The page that the item was detected on
- The location of the item on the document page

You can use synchronous or asynchronous operations to analyze text in a document. To analyze text synchronously, use the AnalyzeDocument (p. 187) operation, and pass a document as input. `AnalyzeDocument` returns the entire set of results. For more information, see Analyzing Document Text with Amazon Textract (p. 90).

To detect text asynchronously, use StartDocumentAnalysis (p. 217) to start processing. To get the results, call GetDocumentAnalysis (p. 201). The results are returned in one or more responses from `GetDocumentAnalysis`. For more information and an example, see Detecting or Analyzing Text in a Multipage Document (p. 119).

To specify which type of analysis to perform, you can use the `FeatureTypes` list input parameter. Add TABLES to the list to return information about the tables that are detected in the input document—for example, table cells, cell text, and selection elements in cells. Add FORMS to return word relationships, such as key-value pairs and selection elements. To perform both types of analysis, add both TABLES and FORMS to `FeatureTypes`.

All lines and words that are detected in the document are included in the response (including text not related to the value of `FeatureTypes`).

Analyzing Invoices and Receipts

Amazon Textract extracts relevant data such as contact information, items purchased, and vendor name, from almost any invoice or receipt without the need for any templates or configuration. Invoices and receipts often use various layouts, making it difficult and time-consuming to manually extract data at scale. Amazon Textract uses ML to understand the context of invoices and receipts and automatically extracts data such as invoice or receipt date, invoice or receipt number, item prices, total amount, and payment terms to suit your business needs.

Amazon Textract also identifies vendor names that are critical for your workflows but may not be explicitly labeled. For example, Amazon Textract can find the vendor name on a receipt even if it's only indicated within a logo at the top of the page without an explicit key-value pair combination. Amazon Textract also makes it easy for you to consolidate input from diverse receipts and invoices that use different words for the same concept. For example, Amazon Textract maps relationships between field names in different documents such as customer no., customer number, and account ID, outputting standard taxonomy as `INVOICE_RECEIPT_ID`. In this case, Amazon Textract represents data consistently across different document types. Fields that do not align with the standard taxonomy are categorized as `OTHER`.

The following is a list of the standard fields that AnalyzeExpense currently supports:

- Vendor Name: `VENDOR_NAME`
- Total: `TOTAL`
- Receiver Address: `RECEIVER_ADDRESS`
- Invoice/Receipt Date: `INVOICE_RECEIPT_DATE`
- Invoice/Receipt ID: `INVOICE_RECEIPT_ID`
- Payment Terms: `PAYMENT_TERMS`
- Subtotal: `SUBTOTAL`
- Due Date: `DUE_DATE`
- Tax: `TAX`
- Invoice Tax Payer ID (SSN/ITIN or EIN): `TAX_PAYER_ID`
- Item Name: `ITEM_NAME`
- Item Price: `PRICE`

- Item Quantity: `QUANTITY`

The AnalyzeExpense API returns the following elements for a given document page:

- The number of receipts or invoices within a page represented as `ExpenseIndex`
- The standardized name for individual fields represented as `Type`
- The actual name of the field as it appears on the document, represented as `LabelDetection`
- The value of the corresponding field represented as `ValueDetection`
- The number of pages within the submitted document represented as `Pages`
- The page number on which the field, value, or line items was detected, represented as `PageNumber`
- The geometry, which includes the bounding box and coordinates location of the individual field, value, or line items on the page, represented as `Geometry`
- The confidence score associated with each piece of data detected on the document, represented as `Confidence`
- The entire row of individual line items purchased, represented as `EXPENSE_ROW`

The following is a portion of the API output for a receipt processed by AnalyzeExpense that shows the Total: $55.64 in the document extracted as standard field `TOTAL`, actual text on the document as “Total”, Confidence Score of “97.1”, Page Number “1”, The total value as “$55.64” and the bounding box and polygon coordinates:

```
{
    "Type": {
        "Text": "TOTAL",
        "Confidence": 99.94717407226562
    },
    "LabelDetection": {
        "Text": "Total:",
        "Geometry": {
            "BoundingBox": {
                "Width": 0.09809663146734238,
                "Height": 0.0234375,
                "Left": 0.36822840571403503,
                "Top": 0.8017578125
            },
            "Polygon": [
                {
                    "X": 0.36822840571403503,
                    "Y": 0.8017578125
                },
                {
                    "X": 0.466325044631958,
                    "Y": 0.8017578125
                },
                {
                    "X": 0.466325044631958,
                    "Y": 0.8251953125
                },
                {
                    "X": 0.36822840571403503,
                    "Y": 0.8251953125
                }
        ]
    },
    "Confidence": 97.10792541503906
},
    "ValueDetection": {
        "Text": "$55.64",
        "Geometry": {
            "BoundingBox": {
                "Width": 0.10395314544439316,
                "Height": 0.0244140625,
                "Left": 0.66837477684021,
                "Top": 0.802734375
            },
            "Polygon": [
                {
                    "X": 0.66837477684021,
                    "Y": 0.802734375
                },
                {
                    "X": 0.7723279595375061,
                    "Y": 0.802734375
                },
                {
                    "X": 0.7723279595375061,
                    "Y": 0.8271484375
                },
                {
                    "X": 0.66837477684021,
                    "Y": 0.8271484375
                }
            ]
        },
    "Confidence": 99.85165405273438
},
"PageNumber": 1
}
```

You can use synchronous operations to analyze an invoice or receipt. To analyze these documents, you use the AnalyzeExpense operation and pass a receipt or invoice to it. `AnalyzeExpense` returns the entire set of results. For more information, see Analyzing Invoices and Receipts with Amazon Textract (p. 98).

To analyze invoices and receipts asynchronously, use `StartExpenseAnalysis` to start processing an input document file. To get the results, call `GetExpenseAnalysis`, which returns the results of your request. For more information and an example, see Processing Documents with Asynchronous Operations (p. 108).

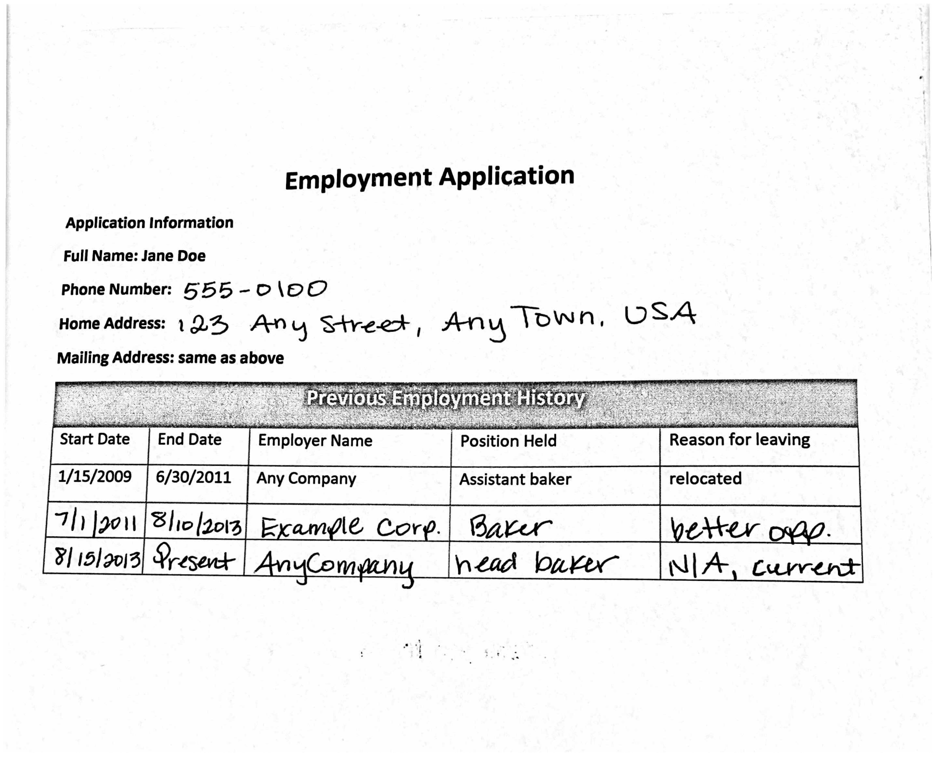
> **Note**
> You cannot use asynchronous operations for AnalyzeExpense. AnalyzeExpense does not support A2I integration. AnalyzeExpense works only on invoices and receipts. For all other documents, use AnalyzeDocument. AnalyzeExpense is only available in our Asia Pacific(Mumbai), Asia Pacific(Seoul), Asia Pacific(Singapore), Asia Pacific(Sydney), Canada(Central), Europe(Frankfurt), Europe(Ireland), Europe(London), U.S. East(N.Virginia), U.S. East(Ohio) U.S. West(N.California), and U.S West(Oregon) regions.

Input Documents

A suitable input for an Amazon Textract operation is a single or multipage document. Some examples are a legal document, a form, or a letter. A form is a document with questions or prompts for a user to provide answers. Some examples are a patient registration form, a tax form, or an insurance claim form.

A document can be in JPEG, PNG, PDF, or TIFF format. Synchronous operations use JPEG, PNG, and TIFF, and Asynchronous operations use all four file types Using PDF and TIFF format files with asynchronous operations enables you to process multipage documents. For information about how Amazon Textract represents documents as `Block` objects, see Text Detection and Document Analysis Response Objects (p. 8).

The following is an acceptable input document example.



Employment Application

Application Information

Full Name: Jane Doe

Phone Number: 555-0100

Home Address: 123 Any Street, Any Town, USA

Mailing Address: same as above

Previous Employment History

Start Date	End Date	Employer Name	Position Held	Reason for leaving
1/15/2009	6/30/2011	Any Company	Assistant baker	relocated
7/1/2011	8/10/2013	Example Corp.	Baker	better opp.
8/15/2013	Present	AnyCompany	head baker	N/A, current

For information about document limits, see Hard Limits in Amazon Textract (p. 255).

For Amazon Textract synchronous operations, you can use input documents that are stored in an Amazon S3 bucket, or you can pass base64-encoded image bytes. For more information, see Calling Amazon Textract Synchronous Operations (p. 32). For asynchronous operations, you need to supply input documents in an Amazon S3 bucket. For more information, see Calling Amazon Textract Asynchronous Operations (p. 108).

Amazon Textract Response Objects

Amazon Textract operations return different types of objects depending on the operations run. For detecting text, and analyzing a generic document, the operation returns a Block object. For analyzing an invoice or receipt, the operation returns an ExpenseDocuments object. For more information about these response objects, see the following sections:

Topics

Text Detection and Document Analysis Response Objects

When Amazon Textract processes a document, it creates a list of Block (p. 231) objects for the detected or analyzed text. Each block contains information about a detected item, where it's located, and the confidence that Amazon Textract has in the accuracy of the processing.

A document is made up from the following types of `Block` objects.

- Pages (p. 10)
- Lines and words of text (p. 11)
- Form Data (Key-value pairs) (p. 13)
- Tables and Cells (p. 15)
- Selection elements (p. 17)

The contents of a block depend on the operation you call. If you call one of the text detection operations, the pages, lines, and words of detected text are returned. For more information, see Detecting Text (p. 3). If you call one of the document analysis operations, information about detected pages, key-value pairs, tables, selection elements, and text is returned. For more information, see Analyzing Documents (p. 4).

Some `Block` object fields are common to both types of processing. For example, each block has a unique identifier.

For examples that show how to use `Block` objects, see Examples (p. 139).

Document Layout

Amazon Textract returns a representation of a document as a list of different types of `Block` objects that are linked in a parent-to-child relationship or a key-value pair. Metadata that provides the number of pages in a document is also returned. The following is the JSON for a typical `Block` object of type `PAGE`.

```
{
    "Blocks": [
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 1.0,
                    "Top": 0.0,
                    "Left": 0.0,
                    "Height": 1.0
                },
                "Polygon": [
                    {
                        "Y": 0.0,
                        "X": 0.0
                    },
                    {
                        "Y": 0.0,
                        "X": 1.0
                    },
                    {
                        "Y": 1.0,
                        "X": 1.0
                    },
                    {
                        "Y": 1.0,
                        "X": 0.0
                    }
                ]
            },
            "Relationships": [
                {
                    "Type": "CHILD",
                    "Ids": [
                        "2602b0a6-20e3-4e6e-9e46-3be57fd0844b",
                        "82aedd57-187f-43dd-9eb1-4f312ca30042",
                        "52be1777-53f7-42f6-a7cf-6d09bdc15a30",
                        "7ca7caa6-00ef-4cda-b1aa-5571dfed1a7c"
                    ]
                }
            ],
            "BlockType": "PAGE",
            "Id": "8136b2dc-37c1-4300-a9da-6ed8b276ea97"
        }.....

    ],
    "DocumentMetadata": {
        "Pages": 1
    }
}
```

A document is made from one or more `PAGE` blocks. Each page contains a list of child blocks for the primary items detected on the page, such as lines of text and tables. For more information, see Pages (p. 10).

You can determine the type of a `Block` object by inspecting the `BlockType` field.

A `Block` object contains a list of related `Block` objects in the `Relationships` field, which is an array of Relationship (p. 252) objects. A `Relationships` array is either of type CHILD or of type VALUE. An array of type CHILD is used to list the items that are children of the current block. For example, if the current block is of type LINE, `Relationships` contains a list of IDs for the WORD blocks that make up the line of text. An array of type VALUE is used to contain key-value pairs. You can determine the type of the relationship by inspecting the `Type` field of the `Relationship` object.

Child blocks don't have information about their parent Block objects.

For examples that show `Block` information, see Processing Documents with Synchronous Operations (p. 32).

Confidence

Amazon Textract operations return the percentage confidence that Amazon Textract has in the accuracy of the detected item. To get the confidence, use the `Confidence` field of the `Block` object. A higher value indicates a higher confidence. Depending on the scenario, detections with a low confidence might need visual confirmation by a human.

Geometry

Amazon Textract operations return location information about the location of detected items on a document page. To get the location, use the `Geometry` field of the `Block` object. For more information, see Item Location on a Document Page (p. 24).

Pages

A document consists of one or more pages. A the section called " Block " (p. 231) object of type `PAGE` exists for each page of the document. A `PAGE` block object contains a list of the child IDs for the lines of text, key-value pairs, and tables that are detected on the document page.

The JSON for a `PAGE` block looks similar to the following.

```
{
    "Geometry": ....
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "2602b0a6-20e3-4e6e-9e46-3be57fd0844b", // Line - Hello, world.
                "82aedd57-187f-43dd-9eb1-4f312ca30042", // Line - How are you?
                "52be1777-53f7-42f6-a7cf-6d09bdc15a30",
                "7ca7caa6-00ef-4cda-b1aa-5571dfed1a7c"
            ]
        }
    ],
    "BlockType": "PAGE",
    "Id": "8136b2dc-37c1-4300-a9da-6ed8b276ea97"  // Page identifier
},
```

If you're using asynchronous operations with a multipage document that's in PDF format, you can determine the page that a block is located on by inspecting the `Page` field of the `Block` object. A scanned image (an image in JPEG, PNG, or TIFF format) is considered to be a single-page document, even if there's more than one document page on the image. Asynchronous operations always return a `Page` value of 1 for scanned images.

The total number of pages is returned in the `Pages` field of `DocumentMetadata`. `DocumentMetadata` is returned with each list of `Block` objects returned by an Amazon Textract operation.

Lines and Words of Text

Detected text that's returned by Amazon Textract operations is returned in a list of the section called " Block " (p. 231) objects. These objects represent lines of text or textual words that are detected on a document page. The following text shows two lines of text that are made from multiple words.

This is text.

In two separate lines.

Detected text is returned in the `Text` field of a `Block` object. The `BlockType` field determines if the text is a line of text (LINE) or a word (WORD). A *WORD* is one or more ISO basic Latin script characters that aren't separated by spaces. A *LINE* is a string of tab-delimited and contiguous words.

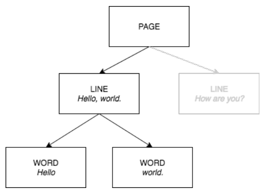
Additionally, Amazon Textract will determine if a piece of text was handwritten or printed using the `TextTypes` field. These return as HANDWRITING and PRINTED respectively.

The other `Block` properties are common to all block types, such as the ID, confidence, and geometry information. For more information, see the section called "Text Detection and Document Analysis Response Objects" (p. 8).

To detect only lines and words, you can use DetectDocumentText (p. 197) or StartDocumentTextDetection (p. 222). For more information, see Detecting Text (p. 3). To get the detected text (lines and words) and information about how it relates to other parts of the document, such as tables, you can use AnalyzeDocument (p. 187) or StartDocumentAnalysis (p. 217). For more information, see Analyzing Documents (p. 4).

`PAGE`, `LINE`, and `WORD` blocks are related to each other in a parent-to-child relationship. A `PAGE` block is the parent for all `LINE` block objects on a document page. Because a LINE can have one or more words, the `Relationships` array for a LINE block stores the IDs for child WORD blocks that make up the line of text.

The following diagram shows how the line *Hello, world.* in the text *Hello, world. How are you?* is represented by `Block` objects.



The following is the JSON output from `DetectDocumentText` when the sentence *Hello, world. How are you?* is detected. The first example is the JSON for the document page. Note how the CHILD IDs enable you to navigate through the document.

```
{
    "Geometry": {...},
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "d7fbd604-d609-4d69-857d-247a3f591238", // Line - Hello, world.
                "b6c19a93-6493-4d8e-958f-853c8f7ca055" //  Line - How are you?
            ]
        }
    ],
    "BlockType": "PAGE",
    "Id": "56ec1d77-171f-4881-9852-2b5b7e761608"
},
```

The following is the JSON for the LINE blocks that make up the line "Hello, World":

```
{
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "7f97e2ca-063e-47a8-981c-8beee31afc01", // Word - Hello,
                "4b990aa0-af96-4369-b90f-dbe02538ed21"  // Word - world.
            ]
        }
    ],
    "Confidence": 99.63229370117188,
    "Geometry": {...},
    "Text": "Hello, world.",
    "BlockType": "LINE",
    "Id": "d7fbd604-d609-4d69-857d-247a3f591238"
},
```

The following is the JSON for the WORD block for the word *Hello,*:

```
{
    "Geometry": {...},
    "Text": "Hello,",
    "TextType": "PRINTED",
    "BlockType": "WORD",
    "Confidence": 99.74746704101562,
    "Id": "7f97e2ca-063e-47a8-981c-8beee31afc01"
},
```

The final JSON is the WORD block for the word *world.*:

```
{
    "Geometry": {...},
    "Text": "world.",
    "TextType": "PRINTED",
    "BlockType": "WORD",
    "Confidence": 99.5171127319336,
    "Id": "4b990aa0-af96-4369-b90f-dbe02538ed21"
},
```

Form Data (Key-Value Pairs)

Amazon Textract can extract form data from documents as key-value pairs. For example, in the following text, Amazon Textract can identify a key (*Name:*) and a value (*Ana Carolina*).

Name: Ana Carolina

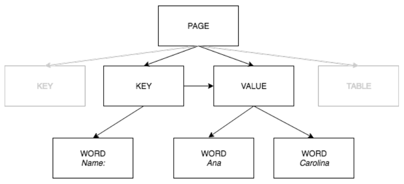
Detected key-value pairs are returned as Block (p. 231) objects in the responses from AnalyzeDocument (p. 187) and GetDocumentAnalysis (p. 201). You can use the `FeatureTypes` input parameter to retrieve information about key-value pairs, tables, or both. For key-value pairs only, use the value `FORMS`. For an example, see Extracting Key-Value Pairs from a Form Document (p. 139). For general information about how a document is represented by `Block` objects, see Text Detection and Document Analysis Response Objects (p. 8).

Block objects with the type KEY_VALUE_SET are the containers for KEY or VALUE Block objects that store information about linked text items detected in a document. You can use the `EntityType` attribute to determine if a block is a KEY or a VALUE.

- A *KEY* object contains information about the key for linked text. For example, *Name:*. A KEY block has two relationship lists. A relationship of type VALUE is a list that contains the ID of the VALUE block associated with the key. A relationship of type CHILD is a list of IDs for the WORD blocks that make up the text of the key.
- A *VALUE* object contains information about the text associated with a key. In the preceding example, *Ana Carolina* is the value for the key *Name:*. A VALUE block has a relationship with a list of CHILD blocks that identify WORD blocks. Each WORD block contains one of the words that make up the text of the value. A `VALUE` object can also contain information about selected elements. For more information, see Selection Elements (p. 17).

Each instance of a KEY_VALUE_SET `Block` object is a child of the PAGE `Block` object that corresponds to the current page.

The following diagram shows how the key-value pair *Name: Ana Carolina* is represented by `Block` objects.



The following examples show how the key-value pair *Name: Ana Carolina* is represented by JSON.

The PAGE block has CHILD blocks of type `KEY_VALUE_SET` for each KEY and VALUE block detected in the document.

```
{
    "Geometry": ....
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "2602b0a6-20e3-4e6e-9e46-3be57fd0844b",
                "82aedd57-187f-43dd-9eb1-4f312ca30042",
                "52be1777-53f7-42f6-a7cf-6d09bdc15a30", // Key - Name:
                "7ca7caa6-00ef-4cda-b1aa-5571dfed1a7c"  // Value - Ana Caroline
            ]
        }
    ],
    "BlockType": "PAGE",
    "Id": "8136b2dc-37c1-4300-a9da-6ed8b276ea97"  // Page identifier
},
```

The following JSON shows that the KEY block (52be1777-53f7-42f6-a7cf-6d09bdc15a30) has a relationship with the VALUE block (7ca7caa6-00ef-4cda-b1aa-5571dfed1a7c). It also has a CHILD block for the WORD block (c734fca6-c4c4-415c-b6c1-30f7510b72ee) that contains the text for the key (*Name:*).

```
{
    "Relationships": [
        {
            "Type": "VALUE",
            "Ids": [
                "7ca7caa6-00ef-4cda-b1aa-5571dfed1a7c"  // Value identifier
            ]
        },
        {
            "Type": "CHILD",
            "Ids": [
                "c734fca6-c4c4-415c-b6c1-30f7510b72ee"  // Name:
            ]
        }
    ],
    "Confidence": 51.55965805053711,
    "Geometry": ....,
    "BlockType": "KEY_VALUE_SET",
    "EntityTypes": [
        "KEY"
    ],
    "Id": "52be1777-53f7-42f6-a7cf-6d09bdc15a30"  //Key identifier
},
```

The following JSON shows that VALUE block 7ca7caa6-00ef-4cda-b1aa-5571dfed1a7c has a CHILD list of IDs for the WORD blocks that make up the text of the value (*Ana* and *Carolina*).

```
{
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "db553509-64ef-4ecf-ad3c-bea62cc1cd8a", // Ana
                "e5d7646c-eaa2-413a-95ad-f4ae19f53ef3"  // Carolina
            ]
        }
    ],
    "Confidence": 51.55965805053711,
    "Geometry": ....,
    "BlockType": "KEY_VALUE_SET",
    "EntityTypes": [
        "VALUE"
    ],
    "Id": "7ca7caa6-00ef-4cda-b1aa-5571dfed1a7c" // Value identifier
}
```

The following JSON shows the `Block` objects for the words *Name:*, *Ana*, and *Carolina*.

```
{
    "Geometry": {...},
    "Text": "Name:",
    "TextType": "PRINTED".
    "BlockType": "WORD",
    "Confidence": 99.56285858154297,
    "Id": "c734fca6-c4c4-415c-b6c1-30f7510b72ee"
},
 {
    "Geometry": {...},
    "Text": "Ana",
    "TextType": "PRINTED",
    "BlockType": "WORD",
    "Confidence": 99.52057647705078,
    "Id": "db553509-64ef-4ecf-ad3c-bea62cc1cd8a"
},
{
    "Geometry": {...},
    "Text": "Carolina",
    "TextType": "PRINTED",
    "BlockType": "WORD",
    "Confidence": 99.84207916259766,
    "Id": "e5d7646c-eaa2-413a-95ad-f4ae19f53ef3"
},
```

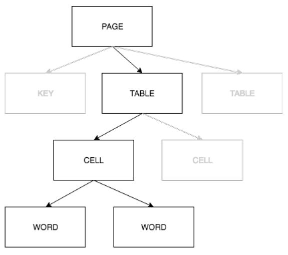
Tables

Amazon Textract can extract tables and the cells in a table. For example, when the following table is detected on a form, Amazon Textract detects a table with four cells.

Name	Address
Ana Carolina	123 Any Town

Detected tables are returned as Block (p. 231) objects in the responses from AnalyzeDocument (p. 187) and GetDocumentAnalysis (p. 201). You can use the `FeatureTypes` input parameter to retrieve information about key-value pairs, tables, or both. For tables only, use the value `TABLES`. For an example, see Exporting Tables into a CSV File (p. 141). For general information about how a document is represented by `Block` objects, see Text Detection and Document Analysis Response Objects (p. 8).

The following diagram shows how a single cell in a table is represented by `Block` objects.



A cell contains `WORD` blocks for detected words, and `SELECTION_ELEMENT` blocks for selection elements such as check boxes.

The following is partial JSON for the preceding table, which has four cells.

The PAGE Block object has a list of CHILD Block IDs for the TABLE block and each LINE of text that's detected.

```
{
    "Geometry": {...},
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "f2a4ad7b-f21d-4966-b548-c859b84f66a4",   // Line - Name
                "4dce3516-ffeb-45e0-92a2-60770e9cb744",   // Line  - Address
                "ee506578-768f-4696-8f4b-e4917e429f50",   // Line - Ana Carolina
                "33fc7223-411b-4399-8a90-ccd3c5a2c196",   // Line  - 123 Any Town
                "3f9665be-379d-4ae7-be44-d02f32b049c2"    // Table
            ]
        }
    ],
    "BlockType": "PAGE",
    "Id": "78c3ce84-ae70-418e-add7-27058418adf6"
},
```

The TABLE block includes a list of child IDs for the cells within the table. A TABLE block also includes geometry information for the table location in the document. The following JSON shows that the table has four cells, which are listed in the `Ids` array.

```
{
    "Geometry": {...},
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "505e9581-0d1c-42fb-a214-6ff736822e8c",
                "6fca44d4-d3d3-46ab-b22f-7fca1fbaaf02",
                "9778bd78-f3fe-4ae1-9b78-e6d29b89e5e9",
                "55404b05-ae12-4159-9003-92b7c129532e"
            ]
        }
    ],
    "BlockType": "TABLE",
    "Confidence": 92.5705337524414,
    "Id": "3f9665be-379d-4ae7-be44-d02f32b049c2"
},
```

The Block type for the table cells is CELL. The `Block` object for each cell includes information about the cell location compared to other cells in the table. It also includes geometry information for the location of the cell on the document. In the preceding example, `505e9581-0d1c-42fb-a214-6ff736822e8c` is the child ID for the cell that contains the word *Name*. The following example is the information for the cell.

```
{
    "Geometry": {...},
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "e9108c8e-0167-4482-989e-8b6cd3c3653e"
            ]
        }
    ],
    "Confidence": 100.0,
    "RowSpan": 1,
    "RowIndex": 1,
    "ColumnIndex": 1,
    "ColumnSpan": 1,
    "BlockType": "CELL",
    "Id": "505e9581-0d1c-42fb-a214-6ff736822e8c"
},
```

Each cell has a location in a table, with the first cell being 1,1. In the preceding example, the cell with the value *Name* is at row 1, column 1. The cell with the value *123 Any Town* is at row 2, column 2. A cell block object contains this information in the `RowIndex` and `ColumnIndex` fields. The child list contains the IDs for the WORD Block objects that contain the text that's within the cell. The words in the list are in the order in which they're detected, from the top left of the cell to the bottom right of the cell. In the preceding example, the cell has a child ID with the value e9108c8e-0167-4482-989e-8b6cd3c3653e. The following output is for the WORD Block with the ID value of e9108c8e-0167-4482-989e-8b6cd3c3653e:

```
"Geometry": {...},
"Text": "Name",
"TextType": "Printed",
"BlockType": "WORD",
"Confidence": 99.81139373779297,
"Id": "e9108c8e-0167-4482-989e-8b6cd3c3653e"
},
```

Selection Elements

Amazon Textract can detect selection elements such as option buttons (radio buttons) and check boxes on a document page. Selection elements can be detected in form data (p. 13) and in tables (p. 15). For example, when the following table is detected on a form, Amazon Textract detects the check boxes in the table cells.

	Agree	**Neutral**	**Disagree**
Good Service	☑	☐	☐
Easy to Use	☐	☑	☐
Fair Price	☑	☐	☐

Detected selection elements are returned as Block (p. 231) objects in the responses from AnalyzeDocument (p. 187) and GetDocumentAnalysis (p. 201).

Note
You can use the `FeatureTypes` input parameter to retrieve information about key-value pairs, tables, or both. For example, if you filter on tables, the response includes the selection elements that are detected in tables. Selection elements that are detected in key-value pairs aren't included in the response.

Information about a selection element is contained in a `Block` object of type `SELECTION_ELEMENT`. To determine the status of a selectable element, use the `SelectionStatus` field of the `SELECTION_ELEMENT` block. The status can be either *SELECTED* or *NOT_SELECTED*. For example, the value of `SelectionStatus` for the previous image is *SELECTED*.

A `SELECTION_ELEMENT Block` object is associated with either a key-value pair or a table cell. A `SELECTION_ELEMENT Block` object contains bounding box information for a selection element in the `Geometry` field. A `SELECTION_ELEMENT Block` object isn't a child of a `PAGE Block` object.

Form Data (Key-Value Pairs)

A key-value pair is used to represent a selection element that's detected on a form. The `KEY` block contains the text for the selection element. The `VALUE` block contains the SELECTION_ELEMENT block.

The following diagram shows how selection elements are represented by the section called " Block " (p. 231) objects.

For more information about key-value pairs, see Form Data (Key-Value Pairs) (p. 13).

The following JSON snippet shows the key for a key-value pair that contains a selection element (**male** ☑). The child ID (Id bd14cfd5-9005-498b-a7f3-45ceb171f0ff) is the ID of the WORD block that contains the text for the selection element (*male*). The value ID (Id 24aaac7f-fcce-49c7-a4f0-3688b05586d4) is the ID of the `VALUE` block that contains the `SELECTION_ELEMENT` block object.

```
{
    "Relationships": [
        {
            "Type": "VALUE",
            "Ids": [
                "24aaac7f-fcce-49c7-a4f0-3688b05586d4"  // Value containing Selection
 Element
            ]
        },
        {
            "Type": "CHILD",
            "Ids": [
                "bd14cfd5-9005-498b-a7f3-45ceb171f0ff"  // WORD - male
            ]
        }
    ],
    "Confidence": 94.15619659423828,
    "Geometry": {
        "BoundingBox": {
            "Width": 0.022914813831448555,
            "Top": 0.08072036504745483,
            "Left": 0.18966935575008392,
            "Height": 0.014860388822853565
        },
        "Polygon": [
            {
                "Y": 0.08072036504745483,
                "X": 0.18966935575008392
            },
            {
                "Y": 0.08072036504745483,
                "X": 0.21258416771888733
            },
            {
                "Y": 0.09558075666427612,
                "X": 0.21258416771888733
            },
            {
                "Y": 0.09558075666427612,
                "X": 0.18966935575008392
            }
        ]
    },
    "BlockType": "KEY_VALUE_SET",
    "EntityTypes": [
        "KEY"
    ],
    "Id": "a118dc43-d5f7-49a2-a20a-5f876d9ffd79"
}
```

The following JSON snippet is the WORD block for the word *Male*. The WORD block also has a parent LINE block.

```
{
    "Geometry": {
        "BoundingBox": {
            "Width": 0.022464623674750328,
            "Top": 0.07842985540628433,
            "Left": 0.18863198161125183,
            "Height": 0.01617223583161831
        },
        "Polygon": [
            {
                "Y": 0.07842985540628433,
                "X": 0.18863198161125183
            },
            {
                "Y": 0.07842985540628433,
                "X": 0.2110965996980667
            },
            {
                "Y": 0.09460209310054779,
                "X": 0.2110965996980667
            },
            {
                "Y": 0.09460209310054779,
                "X": 0.18863198161125183
            }
        ]
    },
    "Text": "Male",
    "BlockType": "WORD",
    "Confidence": 54.06439208984375,
    "Id": "bd14cfd5-9005-498b-a7f3-45ceb171f0ff"
},
```

The VALUE block has a child (Id f2f5e8cd-e73a-4e99-a095-053acd3b6bfb) that is the SELECTION_ELEMENT block.

```
{
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "f2f5e8cd-e73a-4e99-a095-053acd3b6bfb"  // Selection element
            ]
        }
    ],
    "Confidence": 94.15619659423828,
    "Geometry": {
        "BoundingBox": {
            "Width": 0.017281491309404373,
            "Top": 0.07643391191959381,
            "Left": 0.2271782010793686,
            "Height": 0.026274094358086586
        },
        "Polygon": [
            {
                "Y": 0.07643391191959381,
                "X": 0.2271782010793686
            },
            {
                "Y": 0.07643391191959381,
                "X": 0.24445968866348267
            },
            {
                "Y": 0.10270800441503525,
                "X": 0.24445968866348267
            },
            {
                "Y": 0.10270800441503525,
                "X": 0.2271782010793686
            }
        ]
    },
    "BlockType": "KEY_VALUE_SET",
    "EntityTypes": [
        "VALUE"
    ],
    "Id": "24aaac7f-fcce-49c7-a4f0-3688b05586d4"
},
}
```

The following JSON is the SELECTION_ELEMENT block. The value of `SelectionStatus` indicates that the check box is selected.

```
{
    "Geometry": {
        "BoundingBox": {
            "Width": 0.020316146314144135,
            "Top": 0.07575977593660355,
            "Left": 0.22590067982673645,
            "Height": 0.027631107717752457
        },
        "Polygon": [
            {
                "Y": 0.07575977593660355,
                "X": 0.22590067982673645
            },
            {
                "Y": 0.07575977593660355,
                "X": 0.2462168186903
            },
            {
                "Y": 0.1033908873796463,
                "X": 0.2462168186903
            },
            {
                "Y": 0.1033908873796463,
                "X": 0.22590067982673645
            }
        ]
    },
    "BlockType": "SELECTION_ELEMENT",
    "SelectionStatus": "SELECTED",
    "Confidence": 74.14942932128906,
    "Id": "f2f5e8cd-e73a-4e99-a095-053acd3b6bfb"
}
```

Table Cells

Amazon Textract can detect selection elements inside a table cell. For example, the cells in the following table have check boxes.

	Agree	**Neutral**	**Disagree**
Good Service	☑	☐	☐
Easy to Use	☐	☑	☐

Fair Price	☑	☐	☐

A `CELL` block can contain child `SELECTION_ELEMENT` objects for selection elements, as well as child `WORD` blocks for detected text.

For more information about tables, see Tables (p. 15).

The TABLE `Block` object for the previous table looks similar to this.

```
{
    "Geometry": {.....},
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "652c09eb-8945-473d-b1be-fa03ac055928",
                "37efc5cc-946d-42cd-aa04-e68e5ed4741d",
                "4a44940a-435a-4c5c-8a6a-7fea341fa295",
                "2de20014-9a3b-4e26-b453-0de755144b1a",
                "8ed78aeb-5c9a-4980-b669-9e08b28671d2",
                "1f8e1c68-2c97-47b2-847c-a19619c02ca9",
                "9927e1d1-6018-4960-ac17-aadb0a94f4d9",
                "68f0ed8b-a887-42a5-b618-f68b494a6034",
                "fcba16e0-6bd7-4ea5-b86e-36e8330b68ea",
                "2250357c-ae34-4ed9-86da-45dac5a5e903",
                "c63ad40d-5a14-4646-a8df-2d4304213dbc",    // Cell
                "2b8417dc-e65f-4fcd-aa0f-61a23f1e8cb0",
                "26c62932-72f0-4dc2-9893-1ae27829c060",
                "27f291cc-abf4-4c23-aa24-676abe99cb1e",
                "7e5ce028-1bcd-4d9f-ad42-15ac181c5b47",
                "bf32e3d2-efa2-4fc1-b09b-ab9cc52ff734"
            ]
        }
    ],
    "BlockType": "TABLE",
    "Confidence": 99.99993896484375,
    "Id": "f66eac36-2e74-406e-8032-14d1c14e0b86"
}
```

The CELL `BLOCK` object (Id c63ad40d-5a14-4646-a8df-2d4304213dbc) for the cell that contains the check box *Good Service* looks like the following. It includes a child `Block` (Id = 26d122fd-c5f4-4b53-92c4-0ae92730ee1e) that is the `SELECTION_ELEMENT` `Block` object for the check box.

```
{
    "Geometry": {.....},
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "26d122fd-c5f4-4b53-92c4-0ae92730ee1e"   // Selection Element
            ]
        }
    ],
    "Confidence": 79.741689682006836,
    "RowSpan": 1,
    "RowIndex": 3,
    "ColumnIndex": 3,
    "ColumnSpan": 1,
    "BlockType": "CELL",
    "Id": "c63ad40d-5a14-4646-a8df-2d4304213dbc"
}
```

The SELECTION_ELEMENT `Block` object for the check box is as follows. The value of `SelectionStatus` indicates that the check box is selected.

```
{
    "Geometry": {.......},
    "BlockType": "SELECTION_ELEMENT",
    "SelectionStatus": "SELECTED",
    "Confidence": 88.79517364501953,
    "Id": "26d122fd-c5f4-4b53-92c4-0ae92730ee1e"
}
```

Invoice and Receipt Response Objects

When you submit an invoice or a receipt to the AnalyzeExpense API, it returns a series of ExpenseDocuments objects. Each ExpenseDocument is further separated into `LineItemGroups` and `SummaryFields`. Most invoices and receipts contain information such as the vendor name, receipt number, receipt date, or total amount. AnalyzeExpense returns this information under `SummaryFields`. Receipts and invoices also contain details about the items purchased. The AnalyzeExpense API returns this information under `LineItemGroups`. The `ExpenseIndex` field uniquely identifies the expense, and associates the appropriate `SummaryFields` and `LineItemGroups` detected in that expense.

The most granular level of data in the AnalyzeExpense response consists of `Type`, `ValueDetection`, and `LabelDetection` (Optional). The individual entities are:

- Type (p. 22): Refers to what kind of information is detected on a high level.
- LabelDetection (p. 23): Refers to the label of an associated value within the text of the document. `LabelDetection` is optional and only returned if the label is written.
- ValueDetection (p. 23): Refers to the value of the label or type returned.

The AnalyzeExpense API also detects `ITEM`, `QUANTITY`, and `PRICE` within line items as normalized fields. If there is other text in a line item on the receipt image such as SKU or detailed description, it will be included in the JSON as `EXPENSE_ROW` as shown in the below example:

```
                {
                                     "Type": {
                                         "Text": "EXPENSE_ROW",
                                         "Confidence": 99.95216369628906
                                     },
                                     "ValueDetection": {
                                         "Text": "Banana 5 $2.5",
                                         "Geometry": {
                                           …
                                         },
                                         "Confidence": 98.11214447021484
                                     }
```

The example above shows how the AnalyzeExpense API returns the entire row on a receipt that contains line item information about 5 bananas sold for $2.5.

Type

Following is an example of the standard or normalized type of the key-value pair:

```
{
     "PageNumber": 1,
     "Type": {
         "Text": "VENDOR_NAME",
         "Confidence": 70.0
     },
     "ValueDetection": {
         "Geometry": { ... },
         "Text": "AMAZON",
         "Confidence": 87.89806365966797
     }
 }
```

The receipt did not have “Vendor Name” explicitly listed. However, the Analyze Expense API recognized the document as a receipt and categorized the value “AMAZON” as Type `VENDOR_NAME`.

LabelDetection

Following is an example of text as it is shown on a customer document page:

```
{
     "PageNumber": 1,
     "Type": {
         "Text": "OTHER",
         "Confidence": 70.0
     },
     "LabelDetection": {
         "Geometry": { ... },
         "Text": "CASHIER",
         "Confidence": 88.19171142578125
     },
     "ValueDetection": {
         "Geometry": { ... },
         "Text": "Mina",
         "Confidence": 87.89806365966797
     }
 }
```

The example document contained “CASHIER Mina”. The Analyze Expense API extracted the as-is value and returns it under `LabelDetection`. For implied values such as “Vendor Name”, where the “key” is not explicitly shown in the receipt, `LabelDetection` will not be included in the AnalyzeExpense element. In such cases, the AnalyzeExpense API does not return `LabelDetection`.

ValueDetection

The following is an example shows the “value” of the key-value pair.

```
{
     "PageNumber": 1,
     "Type": {
         "Text": "OTHER",
         "Confidence": 70.0
     },
     "LabelDetection": {
         "Geometry": { ... },
         "Text": "CASHIER",
         "Confidence": 88.19171142578125
     },
     "ValueDetection": {
         "Geometry": { ... },
         "Text": "Mina",
         "Confidence": 87.89806365966797
     }
 }
```

In the example, the document contained “CASHIER Mina”. The AnalyzeExpense API detected the Cashier value as Mina and returned it under `ValueDetection`.

Item Location on a Document Page

Amazon Textract operations return the location and geometry of items found on a document page. DetectDocumentText (p. 197) and GetDocumentTextDetection (p. 206) return the location and geometry for lines and words, while AnalyzeDocument (p. 187) and GetDocumentAnalysis (p. 201) return the location and geometry of key-value pairs, tables, cells, and selection elements.

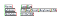
To determine where an item is on a document page, use the bounding box (Geometry (p. 243)) information returned by the Amazon Textract operation in a Block (p. 231) object. The `Geometry` object contains two types of location and geometric information for detected items:

- An axis-aligned BoundingBox (p. 235) object that contains the top-left coordinate and the width and height of the item.
- A polygon object that describes the outline of the item, specified as an array of Point (p. 251) objects that contain `X` (horizontal axis) and `Y` (vertical axis) document page coordinates of each point.

The JSON for a `Block` object looks similar to the following. Note the `BoundingBox` and `Polygon` fields.

```
{
    "Geometry": {
        "BoundingBox": {
            "Width": 0.053907789289951324,
            "Top": 0.08913730084896088,
            "Left": 0.11085548996925354,
            "Height": 0.013171200640499592
        },
        "Polygon": [
            {
                "Y": 0.08985357731580734,
                "X": 0.11085548996925354
            },
            {
                "Y": 0.08913730084896088,
                "X": 0.16447919607162476
            },
            {
                "Y": 0.10159222036600113,
                "X": 0.16476328670978546
            },
            {
                "Y": 0.10230850428342819,
                "X": 0.11113958805799484
            }
        ]
    },
    "Text": "Name:",
    "TextType": "PRINTED",
    "BlockType": "WORD",
    "Confidence": 99.56285858154297,
    "Id": "c734fca6-c4c4-415c-b6c1-30f7510b72ee"
},
```

You can use geometry information to draw bounding boxes around detected items. For an example that uses `BoundingBox` and `Polygon` information to draw boxes around lines and vertical lines at the start and end of each word, see Detecting Document Text with Amazon Textract (p. 81). The example output is similar to the following.



Bounding Box

A bounding box (`BoundingBox`) has the following properties:

- Height – The height of the bounding box as a ratio of the overall document page height.
- Left – The X coordinate of the top-left point of the bounding box as a ratio of the overall document page width.
- Top – The Y coordinate of the top-left point of the bounding box as a ratio of the overall document page height.
- Width – The width of the bounding box as a ratio of the overall document page width.

Each BoundingBox property has a value between 0 and 1. The value is a ratio of the overall image width (applies to `Left` and `Width`) or height (applies to `Height` and `Top`). For example, if the input image is 700 x 200 pixels, and the top-left coordinate of the bounding box is (350,50) pixels, the API returns a `Left` value of 0.5 (350/700) and a `Top` value of 0.25 (50/200).

The following diagram shows the range of a document page that each BoundingBox property covers.

To display the bounding box with the correct location and size, you have to multiply the BoundingBox values by the document page width or height (depending on the value you want) to get the pixel values. You use the pixel values to display the bounding box. An example is using a document page of 608 pixels width x 588 pixels height, and the following bounding box values for analyzed text:

```
BoundingBox.Left: 0.3922065
BoundingBox.Top: 0.15567766
BoundingBox.Width: 0.284666
BoundingBox.Height: 0.2930403
```

The location of the text bounding box in pixels is calculated as follows:

```
Left coordinate = BoundingBox.Left (0.3922065) * document page width (608) =
238

Top coordinate = BoundingBox.Top (0.15567766) * document page height (588) = 91

Bounding box width = BoundingBox.Width (0.284666) * document page width (608) =
173

Bounding box height = BoundingBox.Height (0.2930403) * document page height
(588) = 172
```

You use these values to display a bounding box around the analyzed text. The following Java and Python examples demonstrate how to display a bounding box.

Java

```
   public void ShowBoundingBox(int imageHeight, int imageWidth, BoundingBox box,
Graphics2D g2d) {

       float left = imageWidth * box.getLeft();
       float top = imageHeight * box.getTop();

       // Display bounding box.
       g2d.setColor(new Color(0, 212, 0));
       g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
               Math.round((imageWidth * box.getWidth()) / scale),
Math.round((imageHeight * box.getHeight())) / scale);

   }
```

Python

This Python example takes in the `response` returned by the DetectDocumentText (p. 197) API operation.

```

def process_text_detection(response):

    # Get the text blocks
    blocks = response['Blocks']
    width, height = image.size
    draw = ImageDraw.Draw(image)
    print('Detected Document Text')

    # Create image showing bounding box/polygon the detected lines/text
    for block in blocks:

        draw = ImageDraw.Draw(image)

        if block['BlockType'] == "LINE":
            box=block['Geometry']['BoundingBox']
            left = width * box['Left']
            top = height * box['Top']
            draw.rectangle([left,top, left + (width * box['Width']), top +(height *
 box['Height'])],outline='black')

    # Display the image
    image.show()

    return len(blocks)
```

Polygon

The polygon returned by `AnalyzeDocument` is an array of Point (p. 251) objects. Each `Point` has an X and Y coordinate for a specific location on the document page. Like the BoundingBox coordinates, the polygon coordinates are normalized to the document width and height, and are between 0 and 1.

You can use points in the polygon array to display a finer-grain bounding box around a `Block` object. You calculate the position of each polygon point on the document page by using the same technique used for `BoundingBoxes`. Multiply the X coordinate by the document page width, and multiply the Y coordinate by the document page height.

The following example shows how to display the vertical lines of a polygon.

```
   public void ShowPolygonVerticals(int imageHeight, int imageWidth, List <Point> points,
Graphics2D g2d) {

       g2d.setColor(new Color(0, 212, 0));
       Object[] parry = points.toArray();
       g2d.setStroke(new BasicStroke(2));

       g2d.drawLine(Math.round(((Point) parry[0]).getX() * imageWidth),
               Math.round(((Point) parry[0]).getY() * imageHeight), Math.round(((Point)
parry[3]).getX() * imageWidth),
               Math.round(((Point) parry[3]).getY() * imageHeight));

       g2d.setColor(new Color(255, 0, 0));
       g2d.drawLine(Math.round(((Point) parry[1]).getX() * imageWidth),
               Math.round(((Point) parry[1]).getY() * imageHeight), Math.round(((Point)
parry[2]).getX() * imageWidth),
               Math.round(((Point) parry[2]).getY() * imageHeight));

   }
```

Getting Started with Amazon Textract

This section provides topics to get you started using Amazon Textract. If you're new to Amazon Textract, we recommend that you first review the concepts and terminology in How Amazon Textract Works (p. 3).

You can try the API by using the demonstration in the Amazon Textract console. For more information, see https://console.aws.amazon.com/textract/.

Topics

Step 1: Set Up an AWS Account and Create an IAM User

Before you use Amazon Textract for the first time, complete the following tasks:

1. Sign Up for AWS (p. 28).
2. Create an IAM User (p. 29).

Sign Up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all released services in AWS. You're charged only for the services that you use.

With Amazon Textract, you pay only for the resources you use. For more information about Amazon Textract usage rates, see Amazon Textract pricing. If you're a new AWS customer, you can get started with Amazon Textract for free. For more information, see AWS Free Usage Tier.

If you already have an AWS account, skip to the next task. If you don't have an AWS account, perform the steps in the following procedure to create one.

To create an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Note your AWS account ID because you'll need it for the next task.

Create an IAM User

Services in AWS, such as Amazon Textract, require that you provide credentials when you access them. This is so that the service can determine whether you have permissions to access the resources owned by that service. The console requires your password. You can create access keys for your AWS account to access the AWS CLI or API. However, we don't recommend that you access AWS by using the credentials for your AWS account. Instead, we recommend that you:

- Use AWS Identity and Access Management (IAM) to create an IAM user.
- Add the user to an IAM group with administrative permissions.

You can then access AWS by using a special URL and that IAM user's credentials.

If you signed up for AWS, but you haven't created an IAM user for yourself, you can create one by using the IAM console. Follow the procedure to create an IAM user in your account.

To create an IAM user and sign in to the console

1. Create an IAM user with administrator permissions in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.
2. As the IAM user, sign in to the AWS Management Console by using a special URL. For more information, see How Users Sign In to Your Account in the *IAM User Guide*.

Note
An IAM user with administrator permissions has unrestricted access to the AWS services in your account. The code examples in this guide assume that you have a user with the `AmazonTextractFullAccess` permissions. `AmazonS3ReadOnlyAccess` is required for examples that access documents that are stored in an Amazon S3 bucket. Depending on your security requirements, you might want to use an IAM group that's limited to these permissions. For more information, see Creating IAM Groups.

For more information about IAM, see the following:

- AWS Identity and Access Management (IAM)
- Getting started
- IAM User Guide

Next Step

Step 2: Set Up the AWS CLI and AWS SDKs (p. 29)

Step 2: Set Up the AWS CLI and AWS SDKs

The following steps show you how to install the AWS Command Line Interface (AWS CLI) and AWS SDKs that the examples in this documentation use.

There are a number of different ways to authenticate AWS SDK calls. The examples in this guide assume that you're using a default credentials profile for calling AWS CLI commands and AWS SDK API operations. Your default credentials will work across services, so if you have already configured your credentials you don't need to do so again. However, if you would like to create another set of credentials for this service, you can create a name profile. For more information about creating profiles, see Named Profiles.

For a list of available AWS Regions, see Regions and Endpoints in the *Amazon Web Services General Reference*.

To set up the AWS CLI and the AWS SDKs

1. Download and install the AWS CLI and the AWS SDKs that you want to use. This guide provides examples for the AWS CLI, Java, and Python. For information about other AWS SDKs, see Tools for Amazon Web Services.
   - AWS CLI
   - AWS SDK for Java
   - AWS SDK for Python (Boto3)
2. Create an access key for the user that you created in Create an IAM User (p. 29).
   a. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
   b. In the navigation pane, choose **Users**.
   c. Choose the name of the user that you created in Create an IAM User (p. 29).
   d. Choose the **Security credentials** tab.
   e. Choose **Create access key**. Then choose **Download .csv file** to save the access key ID and secret access key to a CSV file on your computer. Store the file in a secure location. You will not have access to the secret access key again after this dialog box closes. After you've downloaded the CSV file, choose **Close**.
3. Set credentials in the AWS credentials profile file on your local system, located at:
   - `~/.aws/credentials` on Linux, macOS, or Unix.
   - `C:\Users\USERNAME\.aws\credentials` on Windows.

   The `.aws` folder does not exist prior to your first initial configuration of your AWS instance. The first time you configure your credentials with the CLI, this folder will be created. For more information regarding AWS credentials, see Configuration and Credential File Settings.

   This file should contain lines in the following format:

   ```
   [default]
   aws_access_key_id = your_access_key_id
   aws_secret_access_key = your_secret_access_key
   ```

   Substitute your access key ID and secret access key for *your_access_key_id* and *your_secret_access_key*.
4. Set the default AWS Region in the AWS `config` file on your local system, located at:
   - `~/.aws/config` on Linux, macOS, or Unix.
   - `C:\Users\USERNAME\.aws\config` on Windows.

   The `.aws` folder does not exist prior to your first initial configuration of your AWS instance. The first time you configure your credentials with the CLI, this folder will be created. For more information regarding AWS credentials, see Configuration and Credential File Settings.

   This file should contain the following lines:

```
[default]
region = your_aws_region
```

Substitute the AWS Region you want (for example, "us-west-2") for *your_aws_region*.

Note
If you don't choose a Region, then us-east-1 is used by default.

Next Step

Step 3: Get Started Using the AWS CLI and AWS SDK API (p. 31)

Step 3: Get Started Using the AWS CLI and AWS SDK API

After you've set up the AWS CLI and AWS SDKs that you want to use, you can build applications that use Amazon Textract. The following topics show you how to get started with Amazon Textract.

- Analyzing Document Text with Amazon Textract (p. 90)

Formatting the AWS CLI Examples

The AWS CLI examples in this guide are formatted for the Linux operating system. To use the samples with Microsoft Windows, you need to change the JSON formatting of the `--document` parameter, and change the line breaks from backslashes (\) to carets (^). For more information about JSON formatting, see Specifying Parameter Values for the AWS Command Line Interface.

Processing Documents with Synchronous Operations

Amazon Textract can detect and analyze text in single-page documents that are provided as images in JPEG, PNG, and TIFF format. The operations are synchronous and return results in near real time. For more information about documents, see Text Detection and Document Analysis Response Objects (p. 8).

This section covers how you can use Amazon Textract to detect and analyze text in a single-page document. To detect and analyze text in multipage documents or single-page documents that are in PDF or TIFF format, see Processing Documents with Asynchronous Operations (p. 108).

You can use Amazon Textract synchronous operations for the following purposes:

- Text detection – You can detect lines and words on a single-page document image by using the DetectDocumentText (p. 197) operation. For more information, see Detecting Text (p. 3).
- Text analysis – You can identify relationships between detected text on a single-page document by using the AnalyzeDocument (p. 187) operation. For more information, see Analyzing Documents (p. 4).
- Invoice and Receipt Analysis – You can identify financially-related relationships between detected text on a single-page document using the AnalyzeExpense operation. For more information, see Analyzing Invoices and Receipts (p. 5)

Topics

Calling Amazon Textract Synchronous Operations

Amazon Textract operations process document images that are stored on a local file system, or document images stored in an Amazon S3 bucket. You specify where the input document is located by using the Document (p. 236) input parameter. The document image can be in either PNG, JPEG, or TIFF format. Results for synchronous operations are returned immediately and are not stored for retrieval.

For a complete example, see Detecting Document Text with Amazon Textract (p. 81).

Request

The following describes how requests work in Amazon Textract.

Documents Passed as Image Bytes

You can pass a document image to an Amazon Textract operation by passing the image as a base64-encoded byte array. An example is a document image loaded from a local file system. Your code might not need to encode document file bytes if you're using an AWS SDK to call Amazon Textract API operations.

The image bytes are specified in the `Bytes` field of the `Document` input parameter. The following example shows the input JSON for an Amazon Textract operation that passes the image bytes in the `Bytes` input parameter.

```
{
    "Document": {
        "Bytes": "/9j/4AAQSk....."
    }
}
```

> **Note**
> If you're using the AWS CLI, you can't pass image bytes to Amazon Textract operations. Instead, you must reference an image stored in an Amazon S3 bucket.

The following Java code shows how to load an image from a local file system and call an Amazon Textract operation.

```
String document="input.png";

ByteBuffer imageBytes;
try (InputStream inputStream = new FileInputStream(new File(document))) {
    imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
}
AmazonTextract client = AmazonTextractClientBuilder.defaultClient();

DetectDocumentTextRequest request = new DetectDocumentTextRequest()
        .withDocument(new Document()
                .withBytes(imageBytes));


DetectDocumentTextResult result = client.detectDocumentText(request);
```

Documents Stored in an Amazon S3 Bucket

Amazon Textract can analyze document images that are stored in an Amazon S3 bucket. You specify the bucket and file name by using the S3Object (p. 253) field of the `Document` input parameter. The following example shows the input JSON for an Amazon Textract operation that processes a document stored in an Amazon S3 bucket.

```
{
    "Document": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.png"
        }
    }
}
```

The following example shows how to call an Amazon Textract operation using an image stored in an Amazon S3 bucket.

```
String document="input.png";
String bucket="bucket";

AmazonTextract client = AmazonTextractClientBuilder.defaultClient();

DetectDocumentTextRequest request = new DetectDocumentTextRequest()
        .withDocument(new Document()
                .withS3Object(new S3Object()
                        .withName(document)
                        .withBucket(bucket)));

DetectDocumentTextResult result = client.detectDocumentText(request);
```

Response

The following sample is the JSON response from a call to `DetectDocumentText`. For more information, see Detecting Text (p. 3).

```
{
{
  "DocumentMetadata": {
    "Pages": 1
  },
  "Blocks": [
    {
      "BlockType": "PAGE",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.9995205998420715,
          "Height": 1.0,
          "Left": 0.0,
          "Top": 0.0
        },
        "Polygon": [
          {
            "X": 0.0,
            "Y": 0.0
          },
          {
            "X": 0.9995205998420715,
            "Y": 2.297314024515845E-16
          },
          {
            "X": 0.9995205998420715,
            "Y": 1.0
          },
          {
            "X": 0.0,
            "Y": 1.0
          }
        ]
      },
      "Id": "ca4b9171-7109-4adb-a811-e09bbe4834dd",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "26085884-d005-4144-b4c2-4d83dc50739b",
            "ee9d01bc-d91c-401d-8c0a-eec76f5f7862",
            "404bb3d3-d7ab-4008-a195-5dec87a08664",
            "8ae1b4ba-67c1-4486-bd20-54f461886ce9",
            "47aab5ab-be2c-4c73-97c7-d0a45454e843",
            "dd06bb49-6a56-4ea7-beec-a2aa09835c3c",
            "8837153d-81b8-4031-a49f-83a3d81803c2",
            "5dae3b74-9e95-4b62-99b7-93b88fe70648",
            "4508da80-64d8-42a8-8846-cfafe6eab10c",
            "e87be7a9-5519-42e1-b18e-ae10e2d3ed13",
            "f04bb223-d075-41c3-b328-7354611c826b",
            "a234f0e8-67de-46f4-a7c7-0bbe8d5159ce",
            "61b20e27-ff8a-450a-a8b1-bc0259f82fd6",
            "445f4fdd-c77b-4a7b-a2fc-6ca07cfe9ed7",
            "359f3870-7183-43f5-b638-970f5cefe4d5",
            "b9deea0a-244c-4d54-b774-cf03fbaaa8b1",
            "e2a43881-f620-44f2-b067-500ce7dc8d4d",
            "41756974-64ef-432d-b4b2-34702505975a",
            "93d96d32-8b4a-4a98-9578-8b4df4f227a6",
            "bc907357-63d6-43c0-ab87-80d7e76d377e",
            "2d727ca7-3acb-4bb9-a564-5885c90e9325",
            "f32a5989-cbfb-41e6-b0fc-ce1c77c014bd",
            "e0ba06d0-dbb6-4962-8047-8cac3adfe45a",
            "b6ed204d-ae01-4b75-bb91-c85d4147a37e",
            "ac4b9ee0-c9b2-4239-a741-5753e5282033",
            "ebc18885-48d7-45b8-90e3-d172b4357802",
            "babf6360-789e-49c1-9c78-0784acc14a0c"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.93761444091797,
      "Text": "Employment Application",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.3391372561454773,
          "Height": 0.06906412541866302,
          "Left": 0.29548385739326477,
          "Top": 0.027493247762322426
        },
        "Polygon": [
          {
            "X": 0.29548385739326477,
            "Y": 0.027493247762322426
          },
          {
            "X": 0.6346210837364197,
            "Y": 0.027493247762322426
          },
          {
            "X": 0.6346210837364197,
            "Y": 0.0965573713183403
          },
          {
            "X": 0.29548385739326477,
            "Y": 0.0965573713183403
          }
        ]
      },
      "Id": "26085884-d005-4144-b4c2-4d83dc50739b",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "ed48dacc-d089-498f-8e93-1cee1e5f39f3",
            "ac7370f3-cbb7-4cd9-a8f9-bdcb2252caaf"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.91246795654297,
      "Text": "Application Information",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.19878505170345306,
          "Height": 0.03754019737243652,
          "Left": 0.03988289833068848,
          "Top": 0.14050349593162537
        },
        "Polygon": [
          {
            "X": 0.03988289833068848,
            "Y": 0.14050349593162537
          },
          {
            "X": 0.23866795003414154,
            "Y": 0.14050349593162537
          },
          {
            "X": 0.23866795003414154,
            "Y": 0.1780436933040619
          },
          {
            "X": 0.03988289833068848,
            "Y": 0.1780436933040619
          }
        ]
      },
      "Id": "ee9d01bc-d91c-401d-8c0a-eec76f5f7862",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "efe3fc6d-becb-4520-80ee-49a329386aee",
            "c2260852-6cfd-4a71-9fc6-62b2f9b02355"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.88693237304688,
      "Text": "Full Name: Jane Doe",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.16733919084072113,
          "Height": 0.031106337904930115,
          "Left": 0.03899926319718361,
          "Top": 0.21361036598682404
        },
        "Polygon": [
          {
            "X": 0.03899926319718361,
            "Y": 0.21361036598682404
          },
          {
            "X": 0.20633845031261444,
            "Y": 0.21361036598682404
          },
          {
            "X": 0.20633845031261444,
            "Y": 0.24471670389175415
          },
          {
            "X": 0.03899926319718361,
            "Y": 0.24471670389175415
          }
        ]
      },
      "Id": "404bb3d3-d7ab-4008-a195-5dec87a08664",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "e94eb587-9545-4215-b0fc-8e8cb1172958",
            "090aeba5-8428-4b7a-a54b-7a95a774120e",
            "64ff0abb-736b-4a6b-aa8d-ad2c0086ae1d",
            "565ffc30-89d6-4295-b8c6-d22b4ed76584"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.9206314086914,
      "Text": "Phone Number: 555-0100",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.3115004599094391,
          "Height": 0.047169625759124756,
          "Left": 0.03604753687977791,
          "Top": 0.2812676727771759
        },
        "Polygon": [
          {
            "X": 0.03604753687977791,
            "Y": 0.2812676727771759
          },
          {
            "X": 0.3475480079650879,
            "Y": 0.2812676727771759
          },
          {
            "X": 0.3475480079650879,
            "Y": 0.32843729853630066
          },
          {
            "X": 0.03604753687977791,
            "Y": 0.32843729853630066
          }
        ]
      },
      "Id": "8ae1b4ba-67c1-4486-bd20-54f461886ce9",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "d782f847-225b-4a1b-b52d-f252f8221b1f",
            "fa69c5cd-c80d-4fac-81df-569edae8d259",
            "d4bbc0f1-ae02-41cf-a26f-8a1e899968cc"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.48902893066406,
      "Text": "Home Address: 123 Any Street, Any Town. USA",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.7431139945983887,
          "Height": 0.09577702730894089,
          "Left": 0.03359385207295418,
          "Top": 0.3258342146873474
        },
        "Polygon": [
          {
            "X": 0.03359385207295418,
            "Y": 0.3258342146873474
          },
          {
            "X": 0.7767078280448914,
            "Y": 0.3258342146873474
          },
          {
            "X": 0.7767078280448914,
            "Y": 0.4216112196445465
          },
          {
            "X": 0.03359385207295418,
            "Y": 0.4216112196445465
          }
        ]
      },
      "Id": "47aab5ab-be2c-4c73-97c7-d0a45454e843",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "acfbed90-4a00-42c6-8a90-d0a0756eea36",
            "046c8a40-bb0e-4718-9c71-954d3630e1dd",
            "82b838bc-4591-4287-8dea-60c94a4925e4",
            "5cdcde7a-f5a6-4231-a941-b6396e42e7ba",
            "beafd497-185f-487e-b070-db4df5803e94",
            "ef1b77fb-8ba6-41fe-ba53-dce039af22ed",
            "7b555310-e7f8-4cd2-bb3d-dcec37f3d90e",
            "b479c24d-448d-40ef-9ed5-36a6ef08e5c7"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.89382934570312,
      "Text": "Mailing Address: same as above",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.26575741171836853,
          "Height": 0.039571404457092285,
          "Left": 0.03068041242659092,
          "Top": 0.43351811170578003
        },
        "Polygon": [
          {
            "X": 0.03068041242659092,
            "Y": 0.43351811170578003
          },
          {
            "X": 0.2964377999305725,
            "Y": 0.43351811170578003
          },
          {
            "X": 0.2964377999305725,
            "Y": 0.4730895161628723
          },
          {
            "X": 0.03068041242659092,
            "Y": 0.4730895161628723
          }
        ]
      },
      "Id": "dd06bb49-6a56-4ea7-beec-a2aa09835c3c",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "d7261cdc-6ac5-4711-903c-4598fe94952d",
            "287f80c3-6db2-4dd7-90ec-5f017c80aa31",
            "ce31c3ad-b51e-4068-be64-5fc9794bc1bc",
            "e96eb92c-6774-4d6f-8f4a-68a7618d4c66",
            "88b85c05-427a-4d4f-8cc4-3667234e8364"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 94.67343139648438,
      "Text": "Previous Employment History",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.3309842050075531,
          "Height": 0.051920413970947266,
          "Left": 0.3194798231124878,
          "Top": 0.5172380208969116
        },
        "Polygon": [
          {
            "X": 0.3194798231124878,
            "Y": 0.5172380208969116
          },
          {
            "X": 0.6504639983177185,
            "Y": 0.5172380208969116
          },
          {
            "X": 0.6504639983177185,
            "Y": 0.5691584348678589
          },
          {
            "X": 0.3194798231124878,
            "Y": 0.5691584348678589
          }
        ]
      },
      "Id": "8837153d-81b8-4031-a49f-83a3d81803c2",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "8b324501-bf38-4ce9-9777-6514b7ade760",
            "b0cea99a-5045-464d-ac8a-a63ab0470995",
            "b92a6ee5-ca59-44dc-9c47-534c133b11e7"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.66949462890625,
      "Text": "Start Date",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.08310240507125854,
          "Height": 0.030944595113396645,
          "Left": 0.034429505467414856,
          "Top": 0.6123942136764526
        },
        "Polygon": [
          {
              "X": 0.034429505467414856,
              "Y": 0.6123942136764526
            },
            {
              "X": 0.1175319030880928,
              "Y": 0.6123942136764526
            },
            {
              "X": 0.1175319030880928,
              "Y": 0.6433387994766235
            },
            {
              "X": 0.034429505467414856,
              "Y": 0.6433387994766235
            }
          ]
        },
        "Id": "5dae3b74-9e95-4b62-99b7-93b88fe70648",
        "Relationships": [
          {
            "Type": "CHILD",
            "Ids": [
              "ffe8b8e0-df59-4ac5-9aba-6b54b7c51b45",
              "91e582cd-9871-4e9c-93cc-848baa426338"
            ]
          }
        ]
      },
      {
        "BlockType": "LINE",
        "Confidence": 99.86717224121094,
        "Text": "End Date",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.07581500709056854,
            "Height": 0.03223184868693352,
            "Left": 0.14846202731132507,
            "Top": 0.6120467782020569
          },
          "Polygon": [
            {
              "X": 0.14846202731132507,
              "Y": 0.6120467782020569
            },
            {
              "X": 0.22427703440189362,
              "Y": 0.6120467782020569
            },
            {
              "X": 0.22427703440189362,
              "Y": 0.6442786455154419
            },
            {
              "X": 0.14846202731132507,
              "Y": 0.6442786455154419
            }
          ]
        },
        "Id": "4508da80-64d8-42a8-8846-cfafe6eab10c",
        "Relationships": [
          {
            "Type": "CHILD",
            "Ids": [
              "7c97b56b-699f-49b0-93f4-98e6d90b107c",
              "7af04e27-0c15-447e-a569-b30edb99a133"
            ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.9539794921875,
      "Text": "Employer Name",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.1347292959690094,
          "Height": 0.0392492413520813,
          "Left": 0.2647075653076172,
          "Top": 0.6140711903572083
        },
        "Polygon": [
          {
            "X": 0.2647075653076172,
            "Y": 0.6140711903572083
          },
          {
            "X": 0.3994368314743042,
            "Y": 0.6140711903572083
          },
          {
            "X": 0.3994368314743042,
            "Y": 0.6533204317092896
          },
          {
            "X": 0.2647075653076172,
            "Y": 0.6533204317092896
          }
        ]
      },
      "Id": "e87be7a9-5519-42e1-b18e-ae10e2d3ed13",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "a9bfeb55-75cd-47cd-b953-728e602a3564",
            "9f0f9c06-d02c-4b07-bb39-7ade70be2c1b"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.35584259033203,
      "Text": "Position Held",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.11393272876739502,
          "Height": 0.03415105864405632,
          "Left": 0.49973347783088684,
          "Top": 0.614840030670166
        },
        "Polygon": [
          {
            "X": 0.49973347783088684,
            "Y": 0.614840030670166
          },
          {
            "X": 0.6136661767959595,
            "Y": 0.614840030670166
          },
          {
            "X": 0.6136661767959595,
            "Y": 0.6489911079406738
          },
          {
            "X": 0.49973347783088684,
            "Y": 0.6489911079406738
          }
        ]
      },
      "Id": "f04bb223-d075-41c3-b328-7354611c826b",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "6d5edf02-845c-40e0-9514-e56d0d652ae0",
            "3297ab59-b237-45fb-ae60-a108f0c95ac2"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.9817886352539,
      "Text": "Reason for leaving",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.16511960327625275,
          "Height": 0.04062700271606445,
          "Left": 0.7430596351623535,
          "Top": 0.6116235852241516
        },
        "Polygon": [
          {
            "X": 0.7430596351623535,
            "Y": 0.6116235852241516
          },
          {
            "X": 0.9081792235374451,
            "Y": 0.6116235852241516
          },
          {
            "X": 0.9081792235374451,
            "Y": 0.6522505879402161
          },
          {
            "X": 0.7430596351623535,
            "Y": 0.6522505879402161
          }
        ]
      },
      "Id": "a234f0e8-67de-46f4-a7c7-0bbe8d5159ce",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "f4b8cf26-d2da-4a76-8345-69562de3cc11",
            "386d4a63-1194-4c0e-a18d-4d074a0b1f93",
            "a8622541-1896-4d54-8d10-7da2c800ec5c"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.77413177490234,
      "Text": "1/15/2009",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.08799663186073303,
          "Height": 0.03832906484603882,
          "Left": 0.03175082430243492,
          "Top": 0.691371738910675
        },
        "Polygon": [
          {
            "X": 0.03175082430243492,
            "Y": 0.691371738910675
          },
          {
            "X": 0.11974745243787766,
            "Y": 0.691371738910675
          },
          {
            "X": 0.11974745243787766,
            "Y": 0.7297008037567139
          },
          {
            "X": 0.03175082430243492,
            "Y": 0.7297008037567139
          }
        ]
      },
      "Id": "61b20e27-ff8a-450a-a8b1-bc0259f82fd6",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "da7a6482-0964-49a4-bc7d-56942ff3b4e1"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.72286224365234,
      "Text": "6/30/2011",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.08843101561069489,
          "Height": 0.03991425037384033,
          "Left": 0.14642837643623352,
          "Top": 0.6919752955436707
        },
        "Polygon": [
          {
            "X": 0.14642837643623352,
            "Y": 0.6919752955436707
          },
          {
            "X": 0.2348593920469284,
            "Y": 0.6919752955436707
          },
          {
            "X": 0.2348593920469284,
            "Y": 0.731889545917511
          },
          {
            "X": 0.14642837643623352,
            "Y": 0.731889545917511
          }
        ]
      },
      "Id": "445f4fdd-c77b-4a7b-a2fc-6ca07cfe9ed7",
        "Relationships": [
          {
            "Type": "CHILD",
            "Ids": [
              "5a8da66a-ecce-4ee9-a765-a46d6cdc6cde"
            ]
          }
        ]
      },
      {
        "BlockType": "LINE",
        "Confidence": 99.86936950683594,
        "Text": "Any Company",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.11800950765609741,
            "Height": 0.03943679481744766,
            "Left": 0.2626699209213257,
            "Top": 0.6972727179527283
          },
          "Polygon": [
            {
              "X": 0.2626699209213257,
              "Y": 0.6972727179527283
            },
            {
              "X": 0.3806794285774231,
              "Y": 0.6972727179527283
            },
            {
              "X": 0.3806794285774231,
              "Y": 0.736709475517273
            },
            {
              "X": 0.2626699209213257,
              "Y": 0.736709475517273
            }
          ]
        },
        "Id": "359f3870-7183-43f5-b638-970f5cefe4d5",
        "Relationships": [
          {
            "Type": "CHILD",
            "Ids": [
              "77749c2b-aa7f-450e-8dd2-62bcaf253ba2",
              "713bad19-158d-4e3e-b01f-f5707ddb04e5"
            ]
          }
        ]
      },
      {
        "BlockType": "LINE",
        "Confidence": 99.582275390625,
        "Text": "Assistant baker",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.13280922174453735,
            "Height": 0.032666124403476715,
            "Left": 0.49814170598983765,
            "Top": 0.699238657951355
          },
          "Polygon": [
            {
              "X": 0.49814170598983765,
              "Y": 0.699238657951355
            },
            {
              "X": 0.630950927734375,
              "Y": 0.699238657951355
            },
            {
              "X": 0.630950927734375,
              "Y": 0.7319048047065735
            },
            {
              "X": 0.49814170598983765,
              "Y": 0.7319048047065735
            }
          ]
        },
        "Id": "b9deea0a-244c-4d54-b774-cf03fbaaa8b1",
        "Relationships": [
          {
            "Type": "CHILD",
            "Ids": [
              "989944f9-f684-4714-87d8-9ad9a321d65c",
              "ae82e2aa-1601-4e0c-8340-1db7ad0c9a31"
            ]
          }
        ]
      },
      {
        "BlockType": "LINE",
        "Confidence": 99.96180725097656,
        "Text": "relocated",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.08668994903564453,
            "Height": 0.033302485942840576,
            "Left": 0.7426905632019043,
            "Top": 0.6974037289619446
          },
          "Polygon": [
            {
              "X": 0.7426905632019043,
              "Y": 0.6974037289619446
            },
            {
              "X": 0.8293805122375488,
              "Y": 0.6974037289619446
            },
            {
              "X": 0.8293805122375488,
              "Y": 0.7307062149047852
            },
            {
              "X": 0.7426905632019043,
              "Y": 0.7307062149047852
            }
          ]
        },
        "Id": "e2a43881-f620-44f2-b067-500ce7dc8d4d",
        "Relationships": [
          {
            "Type": "CHILD",
            "Ids": [
              "a9cf9a8c-fdaa-413e-9346-5a28a98aebdb"
            ]
          }
        ]
      },
      {
```

```
      "BlockType": "LINE",
      "Confidence": 99.98190307617188,
      "Text": "7/1/2011",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.09747002273797989,
          "Height": 0.07067441940307617,
          "Left": 0.028500309213995934,
          "Top": 0.7745237946510315
        },
        "Polygon": [
          {
            "X": 0.028500309213995934,
            "Y": 0.7745237946510315
          },
          {
            "X": 0.12597033381462097,
            "Y": 0.7745237946510315
          },
          {
            "X": 0.12597033381462097,
            "Y": 0.8451982140541077
          },
          {
            "X": 0.028500309213995934,
            "Y": 0.8451982140541077
          }
        ]
      },
      "Id": "41756974-64ef-432d-b4b2-34702505975a",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "0f711065-1872-442a-ba6d-8fababaa452a"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.98418426513672,
      "Text": "8/10/2013",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.10664612054824829,
          "Height": 0.06439518928527832,
          "Left": 0.14159755408763885,
          "Top": 0.7791688442230225
        },
        "Polygon": [
          {
            "X": 0.14159755408763885,
            "Y": 0.7791688442230225
          },
          {
            "X": 0.24824367463588715,
            "Y": 0.7791688442230225
          },
          {
            "X": 0.24824367463588715,
            "Y": 0.8435640335083008
          },
          {
            "X": 0.14159755408763885,
            "Y": 0.8435640335083008
```

```
                }
              ]
            },
            "Id": "93d96d32-8b4a-4a98-9578-8b4df4f227a6",
            "Relationships": [
              {
                "Type": "CHILD",
                "Ids": [
                  "a92d8eef-db28-45ba-801a-5da0f589d277"
                ]
              }
            ]
          },
          {
            "BlockType": "LINE",
            "Confidence": 99.98075866699219,
            "Text": "Example Corp.",
            "Geometry": {
              "BoundingBox": {
                "Width": 0.2114926278591156,
                "Height": 0.058415766805410385,
                "Left": 0.26764172315597534,
                "Top": 0.794414758682251
              },
              "Polygon": [
                {
                  "X": 0.26764172315597534,
                  "Y": 0.794414758682251
                },
                {
                  "X": 0.47913435101509094,
                  "Y": 0.794414758682251
                },
                {
                  "X": 0.47913435101509094,
                  "Y": 0.8528305292129517
                },
                {
                  "X": 0.26764172315597534,
                  "Y": 0.8528305292129517
                }
              ]
            },
            "Id": "bc907357-63d6-43c0-ab87-80d7e76d377e",
            "Relationships": [
              {
                "Type": "CHILD",
                "Ids": [
                  "d6962efb-34ab-4ffb-9f2f-5f263e813558",
                  "1876c8ea-d3e8-4c39-870e-47512b3b5080"
                ]
              }
            ]
          },
          {
            "BlockType": "LINE",
            "Confidence": 99.91166687011719,
            "Text": "Baker",
            "Geometry": {
              "BoundingBox": {
                "Width": 0.09931200742721558,
                "Height": 0.06008726358413696,
                "Left": 0.5098910331726074,
                "Top": 0.787897527217865
              },
              "Polygon": [
          {
            "X": 0.5098910331726074,
            "Y": 0.787897527217865
          },
          {
            "X": 0.609203040599823,
            "Y": 0.787897527217865
          },
          {
            "X": 0.609203040599823,
            "Y": 0.847984790802002
          },
          {
            "X": 0.5098910331726074,
            "Y": 0.847984790802002
          }
        ]
      },
      "Id": "2d727ca7-3acb-4bb9-a564-5885c90e9325",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "00adeaef-ed57-44eb-b8a9-503575236d62"
          ]
        }
      ]
    },
    {
      "BlockType": "LINE",
      "Confidence": 99.93852233886719,
      "Text": "better opp.",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.18919607996940613,
          "Height": 0.06994765996932983,
          "Left": 0.7428008317947388,
          "Top": 0.7928366661071777
        },
        "Polygon": [
          {
            "X": 0.7428008317947388,
            "Y": 0.7928366661071777
          },
          {
            "X": 0.9319968819618225,
            "Y": 0.7928366661071777
          },
          {
            "X": 0.9319968819618225,
            "Y": 0.8627843260765076
          },
          {
            "X": 0.7428008317947388,
            "Y": 0.8627843260765076
          }
        ]
      },
      "Id": "f32a5989-cbfb-41e6-b0fc-ce1c77c014bd",
      "Relationships": [
        {
          "Type": "CHILD",
          "Ids": [
            "c0fc9a58-7a4b-4f69-bafd-2cff32be2665",
            "bf6dc8ee-2fb3-4b6c-aee4-31e96912a2d8"
          ]
          }
        ]
      },
      {
        "BlockType": "LINE",
        "Confidence": 99.92573547363281,
        "Text": "8/15/2013",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.10257463902235031,
            "Height": 0.05412459373474121,
            "Left": 0.027909137308597565,
            "Top": 0.8608770370483398
          },
          "Polygon": [
            {
              "X": 0.027909137308597565,
              "Y": 0.8608770370483398
            },
            {
              "X": 0.13048377633094788,
              "Y": 0.8608770370483398
            },
            {
              "X": 0.13048377633094788,
              "Y": 0.915001630783081
            },
            {
              "X": 0.027909137308597565,
              "Y": 0.915001630783081
            }
          ]
        },
        "Id": "e0ba06d0-dbb6-4962-8047-8cac3adfe45a",
        "Relationships": [
          {
            "Type": "CHILD",
            "Ids": [
              "5384f860-f857-4a94-9438-9dfa20eed1c6"
            ]
          }
        ]
      },
      {
        "BlockType": "LINE",
        "Confidence": 99.99625396728516,
        "Text": "Present",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.09982697665691376,
            "Height": 0.06888341903686523,
            "Left": 0.1420602649450302,
            "Top": 0.8511748909950256
          },
          "Polygon": [
            {
              "X": 0.1420602649450302,
              "Y": 0.8511748909950256
            },
            {
              "X": 0.24188724160194397,
              "Y": 0.8511748909950256
            },
            {
              "X": 0.24188724160194397,
              "Y": 0.9200583100318909
```

```
                },
                {
                    "X": 0.1420602649450302,
                    "Y": 0.9200583100318909
                }
            ]
        },
        "Id": "b6ed204d-ae01-4b75-bb91-c85d4147a37e",
        "Relationships": [
            {
                "Type": "CHILD",
                "Ids": [
                    "0bb96ed6-b2e6-4da4-90b3-b85561bbd89d"
                ]
            }
        ]
    },
    {
        "BlockType": "LINE",
        "Confidence": 99.9826431274414,
        "Text": "AnyCompany",
        "Geometry": {
            "BoundingBox": {
                "Width": 0.18611276149749756,
                "Height": 0.08581399917602539,
                "Left": 0.2615866959095001,
                "Top": 0.869536280632019
            },
            "Polygon": [
                {
                    "X": 0.2615866959095001,
                    "Y": 0.869536280632019
                },
                {
                    "X": 0.4476994574069977,
                    "Y": 0.869536280632019
                },
                {
                    "X": 0.4476994574069977,
                    "Y": 0.9553502798080444
                },
                {
                    "X": 0.2615866959095001,
                    "Y": 0.9553502798080444
                }
            ]
        },
        "Id": "ac4b9ee0-c9b2-4239-a741-5753e5282033",
        "Relationships": [
            {
                "Type": "CHILD",
                "Ids": [
                    "25343360-d906-440a-88b7-92eb89e95949"
                ]
            }
        ]
    },
    {
        "BlockType": "LINE",
        "Confidence": 99.99549102783203,
        "Text": "head baker",
        "Geometry": {
            "BoundingBox": {
                "Width": 0.1937451809644699,
                "Height": 0.056156039237976074,
                "Left": 0.49359121918678284,
            "Top": 0.8702592849731445
          },
          "Polygon": [
            {
              "X": 0.49359121918678284,
              "Y": 0.8702592849731445
            },
            {
              "X": 0.6873363852500916,
              "Y": 0.8702592849731445
            },
            {
              "X": 0.6873363852500916,
              "Y": 0.9264153242111206
            },
            {
              "X": 0.49359121918678284,
              "Y": 0.9264153242111206
            }
          ]
        },
        "Id": "ebc18885-48d7-45b8-90e3-d172b4357802",
        "Relationships": [
          {
            "Type": "CHILD",
            "Ids": [
              "0ef3c194-8322-4575-94f1-82819ee57e3a",
              "d296acd9-3e9a-4985-95f8-f863614f2c46"
            ]
          }
        ]
      },
      {
        "BlockType": "LINE",
        "Confidence": 99.98360443115234,
        "Text": "N/A, current",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.22544169425964355,
            "Height": 0.06588292121887207,
            "Left": 0.7411766648292542,
            "Top": 0.8722732067108154
          },
          "Polygon": [
            {
              "X": 0.7411766648292542,
              "Y": 0.8722732067108154
            },
            {
              "X": 0.9666183590888977,
              "Y": 0.8722732067108154
            },
            {
              "X": 0.9666183590888977,
              "Y": 0.9381561279296875
            },
            {
              "X": 0.7411766648292542,
              "Y": 0.9381561279296875
            }
          ]
        },
        "Id": "babf6360-789e-49c1-9c78-0784acc14a0c",
        "Relationships": [
          {
            "Type": "CHILD",
          "Ids": [
            "195cfb5b-ae06-4203-8520-4e4b0a73b5ce",
            "549ef3f9-3a13-4b77-bc25-fb2e0996839a"
          ]
        }
      ]
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.94815826416016,
      "Text": "Employment",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.17462396621704102,
          "Height": 0.06266549974679947,
          "Left": 0.29548385739326477,
          "Top": 0.03389188274741173
        },
        "Polygon": [
          {
            "X": 0.29548385739326477,
            "Y": 0.03389188274741173
          },
          {
            "X": 0.4701078236103058,
            "Y": 0.03389188274741173
          },
          {
            "X": 0.4701078236103058,
            "Y": 0.0965573862195015
          },
          {
            "X": 0.29548385739326477,
            "Y": 0.0965573862195015
          }
        ]
      },
      "Id": "ed48dacc-d089-498f-8e93-1cee1e5f39f3"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.92706298828125,
      "Text": "Application",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.15933875739574432,
          "Height": 0.062391020357608795,
          "Left": 0.47528234124183655,
          "Top": 0.027493247762322426
        },
        "Polygon": [
          {
            "X": 0.47528234124183655,
            "Y": 0.027493247762322426
          },
          {
            "X": 0.6346211433410645,
            "Y": 0.027493247762322426
          },
          {
            "X": 0.6346211433410645,
            "Y": 0.0898842737078667
          },
          {
            "X": 0.47528234124183655,
            "Y": 0.08988427370786667
          }
        ]
      },
      "Id": "ac7370f3-cbb7-4cd9-a8f9-bdcb2252caaf"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.9821548461914,
      "Text": "Application",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.09610454738140106,
          "Height": 0.03656719997525215,
          "Left": 0.0398828983306884‌8,
          "Top": 0.14147649705410004
        },
        "Polygon": [
          {
            "X": 0.03988289833068848,
            "Y": 0.14147649705410004
          },
          {
            "X": 0.13598744571208954,
            "Y": 0.14147649705410004
          },
          {
            "X": 0.13598744571208954,
            "Y": 0.1780436933040619
          },
          {
            "X": 0.03988289833068848,
            "Y": 0.1780436933040619
          }
        ]
      },
      "Id": "efe3fc6d-becb-4520-80ee-49a329386aee"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.84278106689453,
      "Text": "Information",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.10029315203428268,
          "Height": 0.03209415823221207,
          "Left": 0.13837480545043945,
          "Top": 0.14050349593162537
        },
        "Polygon": [
          {
            "X": 0.13837480545043945,
            "Y": 0.14050349593162537
          },
          {
            "X": 0.23866795003414154,
            "Y": 0.14050349593162537
          },
          {
            "X": 0.23866795003414154,
            "Y": 0.17259766161441803
          },
          {
            "X": 0.13837480545043945,
            "Y": 0.17259766161441803
          }
        ]
      },
      "Id": "c2260852-6cfd-4a71-9fc6-62b2f9b02355"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.83993530273438,
      "Text": "Full",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.03039788082242012,
          "Height": 0.031106330454349518,
          "Left": 0.03899926319718361,
          "Top": 0.21361036598682404
        },
        "Polygon": [
          {
            "X": 0.03899926319718361,
            "Y": 0.21361036598682404
          },
          {
            "X": 0.06939714401960373,
            "Y": 0.21361036598682404
          },
          {
            "X": 0.06939714401960373,
            "Y": 0.24471670389175415
          },
          {
            "X": 0.03899926319718361,
            "Y": 0.24471670389175415
          }
        ]
      },
      "Id": "e94eb587-9545-4215-b0fc-8e8cb1172958"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.93611907958984,
      "Text": "Name:",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.05555811896920204,
          "Height": 0.030184319242835045,
          "Left": 0.07123806327581406,
          "Top": 0.2137702852487564
        },
        "Polygon": [
          {
            "X": 0.07123806327581406,
            "Y": 0.2137702852487564
          },
          {
            "X": 0.1267961859703064,
            "Y": 0.2137702852487564
          },
          {
            "X": 0.1267961859703064,
            "Y": 0.2439546138048172
          },
          {
            "X": 0.07123806327581406,
            "Y": 0.2439546138048172
          }
        ]
      },
      "Id": "090aeba5-8428-4b7a-a54b-7a95a774120e"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.91043853759766,
      "Text": "Jane",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.03905024006962776,
          "Height": 0.02941947989165783,
          "Left": 0.12933772802352905,
          "Top": 0.214289128780365
        },
        "Polygon": [
          {
            "X": 0.12933772802352905,
            "Y": 0.214289128780365
          },
          {
            "X": 0.16838796436786652,
            "Y": 0.214289128780365
          },
          {
            "X": 0.16838796436786652,
            "Y": 0.24370861053466797
          },
          {
            "X": 0.12933772802352905,
            "Y": 0.24370861053466797
          }
        ]
      },
      "Id": "64ff0abb-736b-4a6b-aa8d-ad2c0086ae1d"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.86123657226562,
      "Text": "Doe",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.035229459404945374,
          "Height": 0.030427640303969383,
          "Left": 0.17110899090766907,
          "Top": 0.21377210319042206
        },
        "Polygon": [
          {
            "X": 0.17110899090766907,
            "Y": 0.21377210319042206
          },
          {
            "X": 0.20633845031261444,
            "Y": 0.21377210319042206
          },
          {
            "X": 0.20633845031261444,
            "Y": 0.244199737906456
          },
          {
            "X": 0.17110899090766907,
            "Y": 0.244199737906456
          }
        ]
      },
      "Id": "565ffc30-89d6-4295-b8c6-d22b4ed76584"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.92633056640625,
      "Text": "Phone",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.052783288061618805,
          "Height": 0.03104414977133274,
          "Left": 0.03604753687977791,
          "Top": 0.28701552748680115
        },
        "Polygon": [
          {
            "X": 0.03604753687977791,
            "Y": 0.28701552748680115
          },
          {
            "X": 0.08883082121610641,
            "Y": 0.28701552748680115
          },
          {
            "X": 0.08883082121610641,
            "Y": 0.31805968284606934
          },
          {
            "X": 0.03604753687977791,
            "Y": 0.31805968284606934
          }
        ]
      },
      "Id": "d782f847-225b-4a1b-b52d-f252f8221b1f"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.86275482177734,
      "Text": "Number:",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.07424934208393097,
          "Height": 0.030300479382276535,
          "Left": 0.0915418416261673,
          "Top": 0.28639692068099976
        },
        "Polygon": [
          {
            "X": 0.0915418416261673,
            "Y": 0.28639692068099976
          },
          {
            "X": 0.16579118371009827,
            "Y": 0.28639692068099976
          },
          {
            "X": 0.16579118371009827,
            "Y": 0.3166973888874054
          },
          {
            "X": 0.0915418416261673,
            "Y": 0.3166973888874054
          }
        ]
      },
      "Id": "fa69c5cd-c80d-4fac-81df-569edae8d259"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.97282409667969,
      "Text": "555-0100",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.17021971940994263,
          "Height": 0.047169629484415054,
          "Left": 0.17732827365398407,
          "Top": 0.2812676727771759
        },
        "Polygon": [
          {
            "X": 0.17732827365398407,
            "Y": 0.2812676727771759
          },
          {
            "X": 0.3475480079650879,
            "Y": 0.2812676727771759
          },
          {
            "X": 0.3475480079650879,
            "Y": 0.32843729853630066
          },
          {
            "X": 0.17732827365398407,
            "Y": 0.32843729853630066
          }
        ]
      },
      "Id": "d4bbc0f1-ae02-41cf-a26f-8a1e899968cc"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.66238403320312,
      "Text": "Home",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.049357783049345016,
          "Height": 0.03134990110993385,
          "Left": 0.03359385207295418,
          "Top": 0.36172014474868774
        },
        "Polygon": [
          {
            "X": 0.03359385207295418,
            "Y": 0.36172014474868774
          },
          {
            "X": 0.0829516351222992,
            "Y": 0.36172014474868774
          },
          {
            "X": 0.0829516351222992,
            "Y": 0.3930700421333313
          },
          {
```

```
          "X": 0.03359385207295418,
          "Y": 0.3930700421333313
        }
      ]
    },
    "Id": "acfbed90-4a00-42c6-8a90-d0a0756eea36"
  },
  {
    "BlockType": "WORD",
    "Confidence": 99.6871109008789,
    "Text": "Address:",
    "TextType": "PRINTED",
    "Geometry": {
      "BoundingBox": {
        "Width": 0.07411003112792969,
        "Height": 0.0314042791724205,
        "Left": 0.08516156673431396,
        "Top": 0.3600046932697296
      },
      "Polygon": [
        {
          "X": 0.08516156673431396,
          "Y": 0.3600046932697296
        },
        {
          "X": 0.15927159786224365,
          "Y": 0.3600046932697296
        },
        {
          "X": 0.15927159786224365,
          "Y": 0.3914089798927307
        },
        {
          "X": 0.08516156673431396,
          "Y": 0.3914089798927307
        }
      ]
    },
    "Id": "046c8a40-bb0e-4718-9c71-954d3630e1dd"
  },
  {
    "BlockType": "WORD",
    "Confidence": 99.93781280517578,
    "Text": "123",
    "TextType": "HANDWRITING",
    "Geometry": {
      "BoundingBox": {
        "Width": 0.05761868134140968,
        "Height": 0.05008566007018089,
        "Left": 0.1750781387090683,
        "Top": 0.35484206676483154
      },
      "Polygon": [
        {
          "X": 0.1750781387090683,
          "Y": 0.35484206676483154
        },
        {
          "X": 0.23269681632518768,
          "Y": 0.35484206676483154
        },
        {
          "X": 0.23269681632518768,
          "Y": 0.40492773056030273
        },
        {
            "X": 0.1750781387090683,
            "Y": 0.40492773056030273
          }
        ]
      },
      "Id": "82b838bc-4591-4287-8dea-60c94a4925e4"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.96530151367188,
      "Text": "Any",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.0681421533226969,
          "Height": 0.06354366987943649,
          "Left": 0.2550157308578491,
          "Top": 0.35471394658088684
        },
        "Polygon": [
          {
            "X": 0.2550157308578491,
            "Y": 0.35471394658088684
          },
          {
            "X": 0.3231579065322876,
            "Y": 0.35471394658088684
          },
          {
            "X": 0.3231579065322876,
            "Y": 0.41825762391090393
          },
          {
            "X": 0.2550157308578491,
            "Y": 0.41825762391090393
          }
        ]
      },
      "Id": "5cdcde7a-f5a6-4231-a941-b6396e42e7ba"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.87527465820312,
      "Text": "Street,",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.12156613171100616,
          "Height": 0.05449587106704712,
          "Left": 0.3357025980949402,
          "Top": 0.3550415635108948
        },
        "Polygon": [
          {
            "X": 0.3357025980949402,
            "Y": 0.3550415635108948
          },
          {
            "X": 0.45726871490478516,
            "Y": 0.3550415635108948
          },
          {
            "X": 0.45726871490478516,
            "Y": 0.4095374345779419
          },
          {
              "X": 0.3357025980949402,
              "Y": 0.4095374345779419
            }
          ]
        },
        "Id": "beafd497-185f-487e-b070-db4df5803e94"
      },
      {
        "BlockType": "WORD",
        "Confidence": 99.99514770507812,
        "Text": "Any",
        "TextType": "HANDWRITING",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.07748188823461533,
            "Height": 0.07339789718389511,
            "Left": 0.47723668813705444,
            "Top": 0.3482133150100708
          },
          "Polygon": [
            {
              "X": 0.47723668813705444,
              "Y": 0.3482133150100708
            },
            {
              "X": 0.554718554019928,
              "Y": 0.3482133150100708
            },
            {
              "X": 0.554718554019928,
              "Y": 0.4216112196445465
            },
            {
              "X": 0.47723668813705444,
              "Y": 0.4216112196445465
            }
          ]
        },
        "Id": "ef1b77fb-8ba6-41fe-ba53-dce039af22ed"
      },
      {
        "BlockType": "WORD",
        "Confidence": 96.80656433105469,
        "Text": "Town.",
        "TextType": "HANDWRITING",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.11213835328817368,
            "Height": 0.057233039289712906,
            "Left": 0.5563329458236694,
            "Top": 0.3331930637359619
          },
          "Polygon": [
            {
              "X": 0.5563329458236694,
              "Y": 0.3331930637359619
            },
            {
              "X": 0.6684713363647461,
              "Y": 0.3331930637359619
            },
            {
              "X": 0.6684713363647461,
              "Y": 0.3904260993003845
            },
            {
                "X": 0.5563329458236694,
                "Y": 0.3904260993003845
              }
            ]
          },
          "Id": "7b555310-e7f8-4cd2-bb3d-dcec37f3d90e"
        },
        {
          "BlockType": "WORD",
          "Confidence": 99.98260498046875,
          "Text": "USA",
          "TextType": "HANDWRITING",
          "Geometry": {
            "BoundingBox": {
              "Width": 0.08771833777427673,
              "Height": 0.05706935003399849,
              "Left": 0.6889894604682922,
              "Top": 0.3258342146873474
            },
            "Polygon": [
              {
                "X": 0.6889894604682922,
                "Y": 0.3258342146873474
              },
              {
                "X": 0.7767078280448914,
                "Y": 0.3258342146873474
              },
              {
                "X": 0.7767078280448914,
                "Y": 0.3829035460948944
              },
              {
                "X": 0.6889894604682922,
                "Y": 0.3829035460948944
              }
            ]
          },
          "Id": "b479c24d-448d-40ef-9ed5-36a6ef08e5c7"
        },
        {
          "BlockType": "WORD",
          "Confidence": 99.9583969116211,
          "Text": "Mailing",
          "TextType": "PRINTED",
          "Geometry": {
            "BoundingBox": {
              "Width": 0.06291338801383972,
              "Height": 0.03957144916057587,
              "Left": 0.03068041242659092,
              "Top": 0.43351811170578003
            },
            "Polygon": [
              {
                "X": 0.03068041242659092,
                "Y": 0.43351811170578003
              },
              {
                "X": 0.09359379857778549,
                "Y": 0.43351811170578003
              },
              {
                "X": 0.09359379857778549,
                "Y": 0.4730895459651947
              },
              {
```

```
            "X": 0.03068041242659092,
            "Y": 0.4730895459651947
          }
        ]
      },
      "Id": "d7261cdc-6ac5-4711-903c-4598fe94952d"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.87476348876953,
      "Text": "Address:",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.07364854216575623,
          "Height": 0.03147412836551666,
          "Left": 0.0954652726650238,
          "Top": 0.43450701236724854
        },
        "Polygon": [
          {
            "X": 0.0954652726650238,
            "Y": 0.43450701236724854
          },
          {
            "X": 0.16911381483078003,
            "Y": 0.43450701236724854
          },
          {
            "X": 0.16911381483078003,
            "Y": 0.465981125831604
          },
          {
            "X": 0.0954652726650238,
            "Y": 0.465981125831604
          }
        ]
      },
      "Id": "287f80c3-6db2-4dd7-90ec-5f017c80aa31"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.94071960449219,
      "Text": "same",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.04640670120716095,
          "Height": 0.026415130123496056,
          "Left": 0.17156922817230225,
          "Top": 0.44010937213897705
        },
        "Polygon": [
          {
            "X": 0.17156922817230225,
            "Y": 0.44010937213897705
          },
          {
            "X": 0.2179759293794632,
            "Y": 0.44010937213897705
          },
          {
            "X": 0.2179759293794632,
            "Y": 0.46652451157569885
          },
          {
            "X": 0.17156922817230225,
            "Y": 0.46652451157569885
          }
        ]
      },
      "Id": "ce31c3ad-b51e-4068-be64-5fc9794bc1bc"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.76510620117188,
      "Text": "as",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.02041218988597393,
          "Height": 0.025104399770498276,
          "Left": 0.2207803726196289,
          "Top": 0.44124215841293335
        },
        "Polygon": [
          {
            "X": 0.2207803726196289,
            "Y": 0.44124215841293335
          },
          {
            "X": 0.24119256436824799,
            "Y": 0.44124215841293335
          },
          {
            "X": 0.24119256436824799,
            "Y": 0.4663465619087219
          },
          {
            "X": 0.2207803726196289,
            "Y": 0.4663465619087219
          }
        ]
      },
      "Id": "e96eb92c-6774-4d6f-8f4a-68a7618d4c66"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.9301528930664,
      "Text": "above",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.05268359184265137,
          "Height": 0.03216424956917763,
          "Left": 0.24375422298908234,
          "Top": 0.4354657828807831
        },
        "Polygon": [
          {
            "X": 0.24375422298908234,
            "Y": 0.4354657828807831
          },
          {
            "X": 0.2964377999305725,
            "Y": 0.4354657828807831
          },
          {
            "X": 0.2964377999305725,
            "Y": 0.4676300287246704
          },
          {
```

```
          "X": 0.24375422298908234,
          "Y": 0.4676300287246704
        }
      ]
    },
    "Id": "88b85c05-427a-4d4f-8cc4-3667234e8364"
  },
  {
    "BlockType": "WORD",
    "Confidence": 85.3905029296875,
    "Text": "Previous",
    "TextType": "PRINTED",
    "Geometry": {
      "BoundingBox": {
        "Width": 0.09860499948263168,
        "Height": 0.04000622034072876,
        "Left": 0.3194798231124878,
        "Top": 0.5194430351257324
      },
      "Polygon": [
        {
          "X": 0.3194798231124878,
          "Y": 0.5194430351257324
        },
        {
          "X": 0.4180848002433777,
          "Y": 0.5194430351257324
        },
        {
          "X": 0.4180848002433777,
          "Y": 0.5594492554664612
        },
        {
          "X": 0.3194798231124878,
          "Y": 0.5594492554664612
        }
      ]
    },
    "Id": "8b324501-bf38-4ce9-9777-6514b7ade760"
  },
  {
    "BlockType": "WORD",
    "Confidence": 99.14524841308594,
    "Text": "Employment",
    "TextType": "PRINTED",
    "Geometry": {
      "BoundingBox": {
        "Width": 0.14039960503578186,
        "Height": 0.04645847901701927,
        "Left": 0.4214291274547577,
        "Top": 0.5219109654426575
      },
      "Polygon": [
        {
          "X": 0.4214291274547577,
          "Y": 0.5219109654426575
        },
        {
          "X": 0.5618287324905396,
          "Y": 0.5219109654426575
        },
        {
          "X": 0.5618287324905396,
          "Y": 0.568369448184967
        },
        {
            "X": 0.4214291274547577,
            "Y": 0.568369448184967
          }
        ]
      },
      "Id": "b0cea99a-5045-464d-ac8a-a63ab0470995"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.48454284667969,
      "Text": "History",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.08361124992370605,
          "Height": 0.05192042887210846,
          "Left": 0.5668527483940125,
          "Top": 0.5172380208969116
        },
        "Polygon": [
          {
            "X": 0.5668527483940125,
            "Y": 0.5172380208969116
          },
          {
            "X": 0.6504639983177185,
            "Y": 0.5172380208969116
          },
          {
            "X": 0.6504639983177185,
            "Y": 0.5691584348678589
          },
          {
            "X": 0.5668527483940125,
            "Y": 0.5691584348678589
          }
        ]
      },
      "Id": "b92a6ee5-ca59-44dc-9c47-534c133b11e7"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.78699493408203,
      "Text": "Start",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.041341401636600494,
          "Height": 0.030926469713449478,
          "Left": 0.034429505467414856,
          "Top": 0.6124123334884644
        },
        "Polygon": [
          {
            "X": 0.034429505467414856,
            "Y": 0.6124123334884644
          },
          {
            "X": 0.07577090710401535,
            "Y": 0.6124123334884644
          },
          {
            "X": 0.07577090710401535,
            "Y": 0.6433387994766235
          },
          {
```

```
          "X": 0.034429505467414856,
          "Y": 0.6433387994766235
        }
      ]
    },
    "Id": "ffe8b8e0-df59-4ac5-9aba-6b54b7c51b45"
  },
  {
    "BlockType": "WORD",
    "Confidence": 99.55198669433594,
    "Text": "Date",
    "TextType": "PRINTED",
    "Geometry": {
      "BoundingBox": {
        "Width": 0.03923053666949272,
        "Height": 0.03072454035282135,
        "Left": 0.07830137014389038,
        "Top": 0.6123942136764526
      },
      "Polygon": [
        {
          "X": 0.07830137014389038,
          "Y": 0.6123942136764526
        },
        {
          "X": 0.1175319105386734,
          "Y": 0.6123942136764526
        },
        {
          "X": 0.1175319105386734,
          "Y": 0.6431187391281128
        },
        {
          "X": 0.07830137014389038,
          "Y": 0.6431187391281128
        }
      ]
    },
    "Id": "91e582cd-9871-4e9c-93cc-848baa426338"
  },
  {
    "BlockType": "WORD",
    "Confidence": 99.8897705078125,
    "Text": "End",
    "TextType": "PRINTED",
    "Geometry": {
      "BoundingBox": {
        "Width": 0.03212086856365204,
        "Height": 0.03193363919854164,
        "Left": 0.14846202731132507,
        "Top": 0.6120467782020569
      },
      "Polygon": [
        {
          "X": 0.14846202731132507,
          "Y": 0.6120467782020569
        },
        {
          "X": 0.1805828958749771,
          "Y": 0.6120467782020569
        },
        {
          "X": 0.1805828958749771,
          "Y": 0.6439804434776306
        },
        {
            "X": 0.14846202731132507,
            "Y": 0.6439804434776306
          }
        ]
      },
      "Id": "7c97b56b-699f-49b0-93f4-98e6d90b107c"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.8445816040039,
      "Text": "Date",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.03987143933773041,
          "Height": 0.03142518177628517,
          "Left": 0.1844055950641632,
          "Top": 0.612853467464447
        },
        "Polygon": [
          {
            "X": 0.1844055950641632,
            "Y": 0.612853467464447
          },
          {
            "X": 0.22427703440189362,
            "Y": 0.612853467464447
          },
          {
            "X": 0.22427703440189362,
            "Y": 0.6442786455154419
          },
          {
            "X": 0.1844055950641632,
            "Y": 0.6442786455154419
          }
        ]
      },
      "Id": "7af04e27-0c15-447e-a569-b30edb99a133"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.9652328491211,
      "Text": "Employer",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.08150768280029297,
          "Height": 0.0392492301762104,
          "Left": 0.2647075653076172,
          "Top": 0.6140711903572083
        },
        "Polygon": [
          {
            "X": 0.2647075653076172,
            "Y": 0.6140711903572083
          },
          {
            "X": 0.34621524810791016,
            "Y": 0.6140711903572083
          },
          {
            "X": 0.34621524810791016,
            "Y": 0.6533204317092896
          },
          {
            "X": 0.2647075653076172,
            "Y": 0.6533204317092896
          }
        ]
      },
      "Id": "a9bfeb55-75cd-47cd-b953-728e602a3564"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.94273376464844,
      "Text": "Name",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.05018233880400658,
          "Height": 0.03248906135559082,
          "Left": 0.34925445914268494,
          "Top": 0.6162016987800598
        },
        "Polygon": [
          {
            "X": 0.34925445914268494,
            "Y": 0.6162016987800598
          },
          {
            "X": 0.3994368016719818,
            "Y": 0.6162016987800598
          },
          {
            "X": 0.3994368016719818,
            "Y": 0.6486907601356506
          },
          {
            "X": 0.34925445914268494,
            "Y": 0.6486907601356506
          }
        ]
      },
      "Id": "9f0f9c06-d02c-4b07-bb39-7ade70be2c1b"
    },
    {
      "BlockType": "WORD",
      "Confidence": 98.85071563720703,
      "Text": "Position",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.07007700204849243,
          "Height": 0.03255689889192581,
          "Left": 0.49973347783088684,
          "Top": 0.6164342164993286
        },
        "Polygon": [
          {
            "X": 0.49973347783088684,
            "Y": 0.6164342164993286
          },
          {
            "X": 0.5698104500770569,
            "Y": 0.6164342164993286
          },
          {
            "X": 0.5698104500770569,
            "Y": 0.6489911079406738
          },
          {
            "X": 0.49973347783088684,
            "Y": 0.6489911079406738
          }
        ]
      },
      "Id": "6d5edf02-845c-40e0-9514-e56d0d652ae0"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.86096954345703,
      "Text": "Held",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.04017873853445053,
          "Height": 0.03292537108063698,
          "Left": 0.5734874606132507,
          "Top": 0.614840030670166
        },
        "Polygon": [
          {
            "X": 0.5734874606132507,
            "Y": 0.614840030670166
          },
          {
            "X": 0.6136662364006042,
            "Y": 0.614840030670166
          },
          {
            "X": 0.6136662364006042,
            "Y": 0.6477653980255127
          },
          {
            "X": 0.5734874606132507,
            "Y": 0.6477653980255127
          }
        ]
      },
      "Id": "3297ab59-b237-45fb-ae60-a108f0c95ac2"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.97740936279297,
      "Text": "Reason",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.06497219949960709,
          "Height": 0.03248770162463188,
          "Left": 0.7430596351623535,
          "Top": 0.6136704087257385
        },
        "Polygon": [
          {
            "X": 0.7430596351623535,
            "Y": 0.6136704087257385
          },
          {
            "X": 0.8080317974090576,
            "Y": 0.6136704087257385
          },
          {
            "X": 0.8080317974090576,
            "Y": 0.6461580991744995
          },
          {
            "X": 0.7430596351623535,
            "Y": 0.6461580991744995
          }
        ]
      },
      "Id": "f4b8cf26-d2da-4a76-8345-69562de3cc11"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.98371887207031,
      "Text": "for",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.029645200818777084,
          "Height": 0.03462234139442444,
          "Left": 0.8108851909637451,
          "Top": 0.6117717623710632
        },
        "Polygon": [
          {
            "X": 0.8108851909637451,
            "Y": 0.6117717623710632
          },
          {
            "X": 0.8405303955078125,
            "Y": 0.6117717623710632
          },
          {
            "X": 0.8405303955078125,
            "Y": 0.6463940739631653
          },
          {
            "X": 0.8108851909637451,
            "Y": 0.6463940739631653
          }
        ]
      },
      "Id": "386d4a63-1194-4c0e-a18d-4d074a0b1f93"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.98424530029297,
      "Text": "leaving",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.06517849862575531,
          "Height": 0.040626998990774155,
          "Left": 0.8430007100105286,
          "Top": 0.6116235852241516
        },
        "Polygon": [
          {
            "X": 0.8430007100105286,
            "Y": 0.6116235852241516
          },
          {
            "X": 0.9081792235374451,
            "Y": 0.6116235852241516
          },
          {
            "X": 0.9081792235374451,
            "Y": 0.6522505879402161
          },
          {
            "X": 0.8430007100105286,
            "Y": 0.6522505879402161
          }
        ]
      },
      "Id": "a8622541-1896-4d54-8d10-7da2c800ec5c"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.77413177490234,
      "Text": "1/15/2009",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.08799663186073303,
          "Height": 0.03832906112074852,
          "Left": 0.03175082430243492,
          "Top": 0.691371738910675
        },
        "Polygon": [
          {
            "X": 0.03175082430243492,
            "Y": 0.691371738910675
          },
          {
            "X": 0.11974745243787766,
            "Y": 0.691371738910675
          },
          {
            "X": 0.11974745243787766,
            "Y": 0.7297008037567139
          },
          {
            "X": 0.03175082430243492,
            "Y": 0.7297008037567139
          }
        ]
      },
      "Id": "da7a6482-0964-49a4-bc7d-56942ff3b4e1"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.72286224365234,
      "Text": "6/30/2011",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.08843102306127548,
          "Height": 0.03991425037384033,
          "Left": 0.14642837643623352,
          "Top": 0.6919752955436707
        },
        "Polygon": [
          {
            "X": 0.14642837643623352,
            "Y": 0.6919752955436707
          },
          {
            "X": 0.2348593920469284,
            "Y": 0.6919752955436707
          },
          {
            "X": 0.2348593920469284,
            "Y": 0.731889545917511
          },
          {
              "X": 0.14642837643623352,
              "Y": 0.731889545917511
            }
          ]
        },
        "Id": "5a8da66a-ecce-4ee9-a765-a46d6cdc6cde"
      },
      {
        "BlockType": "WORD",
        "Confidence": 99.92295837402344,
        "Text": "Any",
        "TextType": "PRINTED",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.034067559987306595,
            "Height": 0.037968240678310394,
            "Left": 0.2626699209213257,
            "Top": 0.6972727179527283
          },
          "Polygon": [
            {
              "X": 0.2626699209213257,
              "Y": 0.6972727179527283
            },
            {
              "X": 0.2967374622821808,
              "Y": 0.6972727179527283
            },
            {
              "X": 0.2967374622821808,
              "Y": 0.7352409362792969
            },
            {
              "X": 0.2626699209213257,
              "Y": 0.7352409362792969
            }
          ]
        },
        "Id": "77749c2b-aa7f-450e-8dd2-62bcaf253ba2"
      },
      {
        "BlockType": "WORD",
        "Confidence": 99.81578063964844,
        "Text": "Company",
        "TextType": "PRINTED",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.08160992711782455,
            "Height": 0.03890080004930496,
            "Left": 0.29906952381134033,
            "Top": 0.6978086829185486
          },
          "Polygon": [
            {
              "X": 0.29906952381134033,
              "Y": 0.6978086829185486
            },
            {
              "X": 0.3806794583797455,
              "Y": 0.6978086829185486
            },
            {
              "X": 0.3806794583797455,
              "Y": 0.736709475517273
            },
            {
```

```
            "X": 0.29906952381134033,
            "Y": 0.736709475517273
          }
        ]
      },
      "Id": "713bad19-158d-4e3e-b01f-f5707ddb04e5"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.37964630126953,
      "Text": "Assistant",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.0789310410618782,
          "Height": 0.031396999995517731,
          "Left": 0.49814170598983765,
          "Top": 0.7005078196525574
        },
        "Polygon": [
          {
            "X": 0.49814170598983765,
            "Y": 0.7005078196525574
          },
          {
            "X": 0.5770727396011353,
            "Y": 0.7005078196525574
          },
          {
            "X": 0.5770727396011353,
            "Y": 0.7319048047065735
          },
          {
            "X": 0.49814170598983765,
            "Y": 0.7319048047065735
          }
        ]
      },
      "Id": "989944f9-f684-4714-87d8-9ad9a321d65c"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.784912109375,
      "Text": "baker",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.050264399498701096,
          "Height": 0.03237773850560188,
          "Left": 0.5806865096092224,
          "Top": 0.699238657951355
        },
        "Polygon": [
          {
            "X": 0.5806865096092224,
            "Y": 0.699238657951355
          },
          {
            "X": 0.630950927734375,
            "Y": 0.699238657951355
          },
          {
            "X": 0.630950927734375,
            "Y": 0.7316163778305054
          },
          {
            "X": 0.5806865096092224,
            "Y": 0.7316163778305054
          }
        ]
      },
      "Id": "ae82e2aa-1601-4e0c-8340-1db7ad0c9a31"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.96180725097656,
      "Text": "relocated",
      "TextType": "PRINTED",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.08668994158506393,
          "Height": 0.03330250084400177,
          "Left": 0.7426905632019043,
          "Top": 0.6974037289619446
        },
        "Polygon": [
          {
            "X": 0.7426905632019043,
            "Y": 0.6974037289619446
          },
          {
            "X": 0.8293805122375488,
            "Y": 0.6974037289619446
          },
          {
            "X": 0.8293805122375488,
            "Y": 0.7307062149047852
          },
          {
            "X": 0.7426905632019043,
            "Y": 0.7307062149047852
          }
        ]
      },
      "Id": "a9cf9a8c-fdaa-413e-9346-5a28a98aebdb"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.98190307617188,
      "Text": "7/1/2011",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.09747002273797989,
          "Height": 0.07067439705133438,
          "Left": 0.028500309213995934,
          "Top": 0.7745237946510315
        },
        "Polygon": [
          {
            "X": 0.028500309213995934,
            "Y": 0.7745237946510315
          },
          {
            "X": 0.12597033381462097,
            "Y": 0.7745237946510315
          },
          {
            "X": 0.12597033381462097,
            "Y": 0.8451982140541077
          },
          {
            "X": 0.028500309213995934,
            "Y": 0.8451982140541077
          }
        ]
      },
      "Id": "0f711065-1872-442a-ba6d-8fababaa452a"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.98418426513672,
      "Text": "8/10/2013",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.10664612054824829,
          "Height": 0.06439515948295593,
          "Left": 0.14159755408763885,
          "Top": 0.7791688442230225
        },
        "Polygon": [
          {
            "X": 0.14159755408763885,
            "Y": 0.7791688442230225
          },
          {
            "X": 0.24824367463588715,
            "Y": 0.7791688442230225
          },
          {
            "X": 0.24824367463588715,
            "Y": 0.843563973903656
          },
          {
            "X": 0.14159755408763885,
            "Y": 0.843563973903656
          }
        ]
      },
      "Id": "a92d8eef-db28-45ba-801a-5da0f589d277"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.97722625732422,
      "Text": "Example",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.12127546221017838,
          "Height": 0.05682983994483948,
          "Left": 0.26764172315597534,
          "Top": 0.794414758682251
        },
        "Polygon": [
          {
            "X": 0.26764172315597534,
            "Y": 0.794414758682251
          },
          {
            "X": 0.3889172077178955,
            "Y": 0.794414758682251
          },
          {
            "X": 0.3889172077178955,
            "Y": 0.8512446284294128
          },
          {
            "X": 0.26764172315597534,
            "Y": 0.8512446284294128
          }
        ]
      },
      "Id": "d6962efb-34ab-4ffb-9f2f-5f263e813558"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.98429870605469,
      "Text": "Corp.",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.07650306820869446,
          "Height": 0.05481306090950966,
          "Left": 0.4026312530040741,
          "Top": 0.7980174422264099
        },
        "Polygon": [
          {
            "X": 0.4026312530040741,
            "Y": 0.7980174422264099
          },
          {
            "X": 0.47913432121276855,
            "Y": 0.7980174422264099
          },
          {
            "X": 0.47913432121276855,
            "Y": 0.8528305292129517
          },
          {
            "X": 0.4026312530040741,
            "Y": 0.8528305292129517
          }
        ]
      },
      "Id": "1876c8ea-d3e8-4c39-870e-47512b3b5080"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.91166687011719,
      "Text": "Baker",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.09931197017431259,
          "Height": 0.06008723005652428,
          "Left": 0.5098910331726074,
          "Top": 0.787897527217865
        },
        "Polygon": [
          {
            "X": 0.5098910331726074,
            "Y": 0.787897527217865
          },
          {
            "X": 0.609203040599823,
            "Y": 0.787897527217865
          },
          {
            "X": 0.609203040599823,
            "Y": 0.8479847311973572
          },
          {
            "X": 0.5098910331726074,
            "Y": 0.8479847311973572
          }
        ]
      },
      "Id": "00adeaef-ed57-44eb-b8a9-503575236d62"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.98870849609375,
      "Text": "better",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.10782185196876526,
          "Height": 0.06207133084535599,
          "Left": 0.7428008317947388,
          "Top": 0.7928366661071777
        },
        "Polygon": [
          {
            "X": 0.7428008317947388,
            "Y": 0.7928366661071777
          },
          {
            "X": 0.8506226539611816,
            "Y": 0.7928366661071777
          },
          {
            "X": 0.8506226539611816,
            "Y": 0.8549079895019531
          },
          {
            "X": 0.7428008317947388,
            "Y": 0.8549079895019531
          }
        ]
      },
      "Id": "c0fc9a58-7a4b-4f69-bafd-2cff32be2665"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.8883285522461,
      "Text": "opp.",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.07421936094760895,
          "Height": 0.058906231075525284,
          "Left": 0.8577775359153748,
          "Top": 0.8038780689239502
        },
        "Polygon": [
          {
            "X": 0.8577775359153748,
            "Y": 0.8038780689239502
          },
          {
            "X": 0.9319969415664673,
            "Y": 0.8038780689239502
          },
          {
            "X": 0.9319969415664673,
            "Y": 0.8627843260765076
          },
          {
            "X": 0.8577775359153748,
            "Y": 0.8627843260765076
          }
        ]
      },
      "Id": "bf6dc8ee-2fb3-4b6c-aee4-31e96912a2d8"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.92573547363281,
      "Text": "8/15/2013",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.10257463902235031,
          "Height": 0.0541245900094509l,
          "Left": 0.027909137308597565,
          "Top": 0.8608770370483398
        },
        "Polygon": [
          {
            "X": 0.027909137308597565,
            "Y": 0.8608770370483398
          },
          {
            "X": 0.13048377633094788,
            "Y": 0.8608770370483398
          },
          {
            "X": 0.13048377633094788,
            "Y": 0.915001630783081
          },
          {
            "X": 0.027909137308597565,
            "Y": 0.915001630783081
          }
        ]
      },
      "Id": "5384f860-f857-4a94-9438-9dfa20eed1c6"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.99625396728516,
      "Text": "Present",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.09982697665691376,
          "Height": 0.06888339668512344,
          "Left": 0.1420602649450302,
          "Top": 0.8511748909950256
        },
        "Polygon": [
          {
            "X": 0.1420602649450302,
            "Y": 0.8511748909950256
          },
          {
            "X": 0.24188724160194397,
            "Y": 0.8511748909950256
          },
          {
            "X": 0.24188724160194397,
            "Y": 0.9200583100318909
          },
          {
            "X": 0.1420602649450302,
            "Y": 0.9200583100318909
          }
        ]
      },
      "Id": "0bb96ed6-b2e6-4da4-90b3-b85561bbd89d"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.9826431274414,
      "Text": "AnyCompany",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.18611273169517517,
          "Height": 0.08581399917602539,
          "Left": 0.2615866959095001,
          "Top": 0.869536280632019
        },
        "Polygon": [
          {
            "X": 0.2615866959095001,
            "Y": 0.869536280632019
          },
          {
            "X": 0.4476994276046753,
            "Y": 0.869536280632019
          },
          {
            "X": 0.4476994276046753,
            "Y": 0.9553502798080444
          },
          {
            "X": 0.2615866959095001,
            "Y": 0.9553502798080444
          }
        ]
      },
      "Id": "25343360-d906-440a-88b7-92eb89e95949"
    },
    {
      "BlockType": "WORD",
      "Confidence": 99.99523162841797,
      "Text": "head",
      "TextType": "HANDWRITING",
      "Geometry": {
        "BoundingBox": {
          "Width": 0.07429949939250946,
          "Height": 0.05485520139336586,
          "Left": 0.49359121918678284,
          "Top": 0.8714361190795898
        },
        "Polygon": [
          {
            "X": 0.49359121918678284,
            "Y": 0.8714361190795898
          },
          {
            "X": 0.5678907036781311,
            "Y": 0.8714361190795898
          },
          {
            "X": 0.5678907036781311,
            "Y": 0.926291286945343
          },
          {
              "X": 0.49359121918678284,
              "Y": 0.926291286945343
            }
          ]
        },
        "Id": "0ef3c194-8322-4575-94f1-82819ee57e3a"
      },
      {
        "BlockType": "WORD",
        "Confidence": 99.99574279785156,
        "Text": "baker",
        "TextType": "HANDWRITING",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.1019822508096695,
            "Height": 0.05615599825978279,
            "Left": 0.585354208946228,
            "Top": 0.8702592849731445
          },
          "Polygon": [
            {
              "X": 0.585354208946228,
              "Y": 0.8702592849731445
            },
            {
              "X": 0.6873364448547363,
              "Y": 0.8702592849731445
            },
            {
              "X": 0.6873364448547363,
              "Y": 0.9264153242111206
            },
            {
              "X": 0.585354208946228,
              "Y": 0.9264153242111206
            }
          ]
        },
        "Id": "d296acd9-3e9a-4985-95f8-f863614f2c46"
      },
      {
        "BlockType": "WORD",
        "Confidence": 99.9880599975586,
        "Text": "N/A,",
        "TextType": "HANDWRITING",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.08230073750019073,
            "Height": 0.06588289886713028,
            "Left": 0.7411766648292542,
            "Top": 0.8722732067108154
          },
          "Polygon": [
            {
              "X": 0.7411766648292542,
              "Y": 0.8722732067108154
            },
            {
              "X": 0.8234773874282837,
              "Y": 0.8722732067108154
            },
            {
              "X": 0.8234773874282837,
              "Y": 0.9381561279296875
            },
            {
              "X": 0.7411766648292542,
              "Y": 0.9381561279296875
            }
          ]
        },
        "Id": "195cfb5b-ae06-4203-8520-4e4b0a73b5ce"
      },
      {
        "BlockType": "WORD",
        "Confidence": 99.97914123535156,
        "Text": "current",
        "TextType": "HANDWRITING",
        "Geometry": {
          "BoundingBox": {
            "Width": 0.12791454792022705,
            "Height": 0.04768490046262741,
            "Left": 0.8387037515640259,
            "Top": 0.8843405842781067
          },
          "Polygon": [
            {
              "X": 0.8387037515640259,
              "Y": 0.8843405842781067
            },
            {
              "X": 0.9666182994842529,
              "Y": 0.8843405842781067
            },
            {
              "X": 0.9666182994842529,
              "Y": 0.9320254921913147
            },
            {
              "X": 0.8387037515640259,
              "Y": 0.9320254921913147
            }
          ]
        },
        "Id": "549ef3f9-3a13-4b77-bc25-fb2e0996839a"
      }
    ],
    "DetectDocumentTextModelVersion": "1.0",
    "ResponseMetadata": {
      "RequestId": "337129e6-3af7-4014-842b-f6484e82cbf6",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
        "x-amzn-requestid": "337129e6-3af7-4014-842b-f6484e82cbf6",
        "content-type": "application/x-amz-json-1.1",
        "content-length": "45675",
        "date": "Mon, 09 Nov 2020 23:54:38 GMT"
      },
      "RetryAttempts": 0
    }
  }
}
```

Detecting Document Text with Amazon Textract

To detect text in a document, you use the DetectDocumentText (p. 197) operation, and pass a document file as input. `DetectDocumentText` returns a JSON structure that contains lines and words of detected text, the location of the text in the document, and the relationships between detected text. For more information, see Detecting Text (p. 3).

You can provide an input document as an image byte array (base64-encoded image bytes), or as an Amazon S3 object. In this procedure, you upload an image file to your S3 bucket and specify the file name.

To detect text in a document (API)

1. If you haven't already:

   a. Create or update an IAM user with `AmazonTextractFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 29).

   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 29).

2. Upload a document to your S3 bucket.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service User Guide*.

3. Use the following examples to call the `DetectDocumentText` operation.

   Java

   The following example code displays the document and boxes around lines of detected text.

   In the function `main`, replace the values of `bucket` and `document` with the names of the Amazon S3 bucket and document that you used in step 2.

```
//Calls DetectDocumentText.
//Loads document from S3 bucket. Displays the document and bounding boxes around
 detected lines/words of text.
package com.amazonaws.samples;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3ObjectInputStream;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.services.textract.AmazonTextract;
import com.amazonaws.services.textract.AmazonTextractClientBuilder;
import com.amazonaws.services.textract.model.Block;
import com.amazonaws.services.textract.model.BoundingBox;
import com.amazonaws.services.textract.model.DetectDocumentTextRequest;
import com.amazonaws.services.textract.model.DetectDocumentTextResult;
import com.amazonaws.services.textract.model.Document;
import com.amazonaws.services.textract.model.S3Object;
import com.amazonaws.services.textract.model.Point;
import com.amazonaws.services.textract.model.Relationship;

public class DocumentText extends JPanel {

    private static final long serialVersionUID = 1L;

    BufferedImage image;
    DetectDocumentTextResult result;

    public DocumentText(DetectDocumentTextResult documentResult, BufferedImage
 bufImage) throws Exception {
        super();
        result = documentResult; // Results of text detection.
        image = bufImage; // The image containing the document.

    }

    // Draws the image and text bounding box.
    public void paintComponent(Graphics g) {

        int height = image.getHeight(this);
        int width = image.getWidth(this);

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, image.getWidth(this) , image.getHeight(this),
this);

        // Iterate through blocks and display polygons around lines of detected
text.
        List<Block> blocks = result.getBlocks();
        for (Block block : blocks) {
            DisplayBlockInfo(block);
            if ((block.getBlockType()).equals("LINE")) {
                ShowPolygon(height, width, block.getGeometry().getPolygon(), g2d);
                /*
                  ShowBoundingBox(height, width,
block.getGeometry().getBoundingBox(), g2d);
                 */
            } else { // its a word, so just show vertical lines.
                ShowPolygonVerticals(height, width,
block.getGeometry().getPolygon(), g2d);
            }
        }
    }

    // Show bounding box at supplied location.
    private void ShowBoundingBox(int imageHeight, int imageWidth, BoundingBox box,
Graphics2D g2d) {

        float left = imageWidth * box.getLeft();
        float top = imageHeight * box.getTop();

        // Display bounding box.
        g2d.setColor(new Color(0, 212, 0));
        g2d.drawRect(Math.round(left), Math.round(top),
                Math.round(imageWidth * box.getWidth()), Math.round(imageHeight *
box.getHeight()));

    }

    // Shows polygon at supplied location
    private void ShowPolygon(int imageHeight, int imageWidth, List<Point> points,
Graphics2D g2d) {

        g2d.setColor(new Color(0, 0, 0));
        Polygon polygon = new Polygon();

        // Construct polygon and display
        for (Point point : points) {
            polygon.addPoint((Math.round(point.getX() * imageWidth)),
                    Math.round(point.getY() * imageHeight));
        }
        g2d.drawPolygon(polygon);
    }

    // Draws only the vertical lines in the supplied polygon.
```

```
   private void ShowPolygonVerticals(int imageHeight, int imageWidth, List<Point>
points, Graphics2D g2d) {

       g2d.setColor(new Color(0, 212, 0));
       Object[] parry = points.toArray();
       g2d.setStroke(new BasicStroke(2));

       g2d.drawLine(Math.round(((Point) parry[0]).getX() * imageWidth),
               Math.round(((Point) parry[0]).getY() * imageHeight),
Math.round(((Point) parry[3]).getX() * imageWidth),
               Math.round(((Point) parry[3]).getY() * imageHeight));

       g2d.setColor(new Color(255, 0, 0));
       g2d.drawLine(Math.round(((Point) parry[1]).getX() * imageWidth),
               Math.round(((Point) parry[1]).getY() * imageHeight),
Math.round(((Point) parry[2]).getX() * imageWidth),
               Math.round(((Point) parry[2]).getY() * imageHeight));

   }
   //Displays information from a block returned by text detection and text
analysis
   private void DisplayBlockInfo(Block block) {
       System.out.println("Block Id : " + block.getId());
       if (block.getText()!=null)
           System.out.println("    Detected text: " + block.getText());
       System.out.println("    Type: " + block.getBlockType());

       if (block.getBlockType().equals("PAGE") !=true) {
           System.out.println("    Confidence: " +
block.getConfidence().toString());
       }
       if(block.getBlockType().equals("CELL"))
       {
           System.out.println("    Cell information:");
           System.out.println("        Column: " + block.getColumnIndex());
           System.out.println("        Row: " + block.getRowIndex());
           System.out.println("        Column span: " + block.getColumnSpan());
           System.out.println("        Row span: " + block.getRowSpan());

       }

       System.out.println("    Relationships");
       List<Relationship> relationships=block.getRelationships();
       if(relationships!=null) {
           for (Relationship relationship : relationships) {
               System.out.println("        Type: " + relationship.getType());
               System.out.println("        IDs: " +
relationship.getIds().toString());
           }
       } else {
           System.out.println("        No related Blocks");
       }

       System.out.println("    Geometry");
       System.out.println("        Bounding Box: " +
block.getGeometry().getBoundingBox().toString());
       System.out.println("        Polygon: " +
block.getGeometry().getPolygon().toString());

       List<String> entityTypes = block.getEntityTypes();

       System.out.println("    Entity Types");
       if(entityTypes!=null) {
           for (String entityType : entityTypes) {
               System.out.println("        Entity Type: " + entityType);
           }
```

```
        } else {
            System.out.println("        No entity type");
        }
        if(block.getPage()!=null)
            System.out.println("    Page: " + block.getPage());
        System.out.println();
    }

    public static void main(String arg[]) throws Exception {

        // The S3 bucket and document
        String document = "";
        String bucket = "";


        AmazonS3 s3client = AmazonS3ClientBuilder.standard()
                .withEndpointConfiguration(
                        new EndpointConfiguration("https://s3.amazonaws.com","us-
east-1"))
                .build();


        // Get the document from S3
        com.amazonaws.services.s3.model.S3Object s3object =
 s3client.getObject(bucket, document);
        S3ObjectInputStream inputStream = s3object.getObjectContent();
        BufferedImage image = ImageIO.read(inputStream);

        // Call DetectDocumentText
        EndpointConfiguration endpoint = new EndpointConfiguration(
                "https://textract.us-east-1.amazonaws.com", "us-east-1");
        AmazonTextract client = AmazonTextractClientBuilder.standard()
                .withEndpointConfiguration(endpoint).build();


        DetectDocumentTextRequest request = new DetectDocumentTextRequest()
            .withDocument(new Document().withS3Object(new
 S3Object().withName(document).withBucket(bucket)));

        DetectDocumentTextResult result = client.detectDocumentText(request);

        // Create frame and panel.
        JFrame frame = new JFrame("RotateImage");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        DocumentText panel = new DocumentText(result, image);
        panel.setPreferredSize(new Dimension(image.getWidth() ,
 image.getHeight() ));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    }
}
```

AWS CLI

This AWS CLI command displays the JSON output for the `detect-document-text` CLI operation.

Replace the values of `Bucket` and `Name` with the names of the Amazon S3 bucket and document that you used in step 2.

```
aws textract detect-document-text \
--document '{"S3Object":{"Bucket":"bucket","Name":"document"}}'
```

Python

The following example code displays the document and boxes around detected lines of text.

In the function `main`, replace the values of `bucket` and `document` with the names of the Amazon S3 bucket and document that you used in step 2.

```
#Detects text in a document stored in an S3 bucket. Display polygon box around text
 and angled text
import boto3
import io
from io import BytesIO
import sys

import psutil
import time

import math
from PIL import Image, ImageDraw, ImageFont


# Displays information about a block returned by text detection and text analysis
def DisplayBlockInformation(block):
    print('Id: {}'.format(block['Id']))
    if 'Text' in block:
        print('    Detected: ' + block['Text'])
    print('    Type: ' + block['BlockType'])

    if 'Confidence' in block:
        print('    Confidence: ' + "{:.2f}".format(block['Confidence']) + "%")

    if block['BlockType'] == 'CELL':
        print("    Cell information")
        print("        Column: " + str(block['ColumnIndex']))
        print("        Row: " + str(block['RowIndex']))
        print("        ColumnSpan: " + str(block['ColumnSpan']))
        print("        RowSpan: " + str(block['RowSpan']))

    if 'Relationships' in block:
        print('    Relationships: {}'.format(block['Relationships']))
    print('    Geometry: ')
    print('        Bounding Box: {}'.format(block['Geometry']['BoundingBox']))
    print('        Polygon: {}'.format(block['Geometry']['Polygon']))

    if block['BlockType'] == "KEY_VALUE_SET":
        print ('    Entity Type: ' + block['EntityTypes'][0])
    if 'Page' in block:
        print('Page: ' + block['Page'])
    print()

def process_text_detection(bucket, document):


    #Get the document from S3
    s3_connection = boto3.resource('s3')

    s3_object = s3_connection.Object(bucket,document)
    s3_response = s3_object.get()

    stream = io.BytesIO(s3_response['Body'].read())
    image=Image.open(stream)
    # Detect text in the document

    client = boto3.client('textract')
    #process using image bytes
    #image_binary = stream.getvalue()
    #response = client.detect_document_text(Document={'Bytes': image_binary})

    #process using S3 object
    response = client.detect_document_text(
        Document={'S3Object': {'Bucket': bucket, 'Name': document}})

    #Get the text blocks
    blocks=response['Blocks']
    width, height =image.size
    draw = ImageDraw.Draw(image)
    print ('Detected Document Text')

    # Create image showing bounding box/polygon the detected lines/text
    for block in blocks:
            print('Type: ' + block['BlockType'])
            if block['BlockType'] != 'PAGE':
                print('Detected: ' + block['Text'])
                print('Confidence: ' + "{:.2f}".format(block['Confidence']) + "%")

            print('Id: {}'.format(block['Id']))
            if 'Relationships' in block:
                print('Relationships: {}'.format(block['Relationships']))
            print('Bounding Box: {}'.format(block['Geometry']['BoundingBox']))
            print('Polygon: {}'.format(block['Geometry']['Polygon']))
            print()
            draw=ImageDraw.Draw(image)
            # Draw WORD - Green -  start of word, red - end of word
            if block['BlockType'] == "WORD":
                draw.line([(width * block['Geometry']['Polygon'][0]['X'],
                height * block['Geometry']['Polygon'][0]['Y']),
                (width * block['Geometry']['Polygon'][3]['X'],
                height * block['Geometry']['Polygon'][3]['Y'])],fill='green',
                width=2)

                draw.line([(width * block['Geometry']['Polygon'][1]['X'],
                height * block['Geometry']['Polygon'][1]['Y']),
                (width * block['Geometry']['Polygon'][2]['X'],
                height * block['Geometry']['Polygon'][2]['Y'])],
                fill='red',
                width=2)


            # Draw box around entire LINE
            if block['BlockType'] == "LINE":
                points=[]

                for polygon in block['Geometry']['Polygon']:
                    points.append((width * polygon['X'], height * polygon['Y']))

                draw.polygon((points), outline='black')

                # Uncomment to draw bounding box
                #box=block['Geometry']['BoundingBox']
                #left = width * box['Left']
                #top = height * box['Top']
                #draw.rectangle([left,top, left + (width * box['Width']), top
+(height * box['Height'])],outline='black')


    # Display the image
    image.show()
    # display image for 10 seconds


    return len(blocks)

def main():

    bucket = ''
    document = ''
    block_count=process_text_detection(bucket,document)
    print("Blocks detected: " + str(block_count))

if __name__ == "__main__":
    main()
```

Node.js

The following Node.js example code displays the document and boxes around detected lines of text, outputting an image of the results to the directory you run the code from. It makes use of the `image-size` and `images` packages.

In the function `main`, replace the values of `bucket` and `document` with the names of the Amazon S3 bucket and document that you used in step 2. Replace the value of `regionConfig` with the name of the region your account is in.

```
async function main(){

// Import AWS
const AWS = require("aws-sdk")
// Use Image-Size to get
const sizeOf = require('image-size');
// Image tool to draw buffers
const images = require("images");

// Create a canvas and get the context
const { createCanvas } = require('canvas')
const canvas = createCanvas(200, 200)
const ctx = canvas.getContext('2d')

// Set variables
const bucket = 'bucket-name' // the s3 bucket name
const photo  = 'image-name' // the name of file
const regionConfig = 'region'

// Set region if needed
AWS.config.update({region:regionConfig});

// Connect to Textract
const client = new AWS.Textract();
// Connect to S3 to display image
const s3 = new AWS.S3();

// Define paramaters
const params = {
  Document: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
}
// Function to display image
async function getImage(){
  const imageData =  s3.getObject(
    {
        Bucket: bucket,
        Key: photo
      }

  ).promise();
  return imageData;
}

// get image
var imageData = await getImage()

// Get the height, width of the image
const dimensions = sizeOf(imageData.Body)
const width = dimensions.width
const height = dimensions.height
console.log(imageData.Body)
console.log(width, height)

canvas.width = width;
canvas.height = height;

try{
  // Call API and log response
  const res = await client.detectDocumentText(params).promise();
  var image = images(imageData.Body).size(width, height)
  //console.log the type of block, text, text type, and confidence
  res.Blocks.forEach(block => {
    console.log(`Block Type: ${block.BlockType}`),
    console.log(`Text: ${block.Text}`)
    console.log(`TextType: ${block.TextType}`)
    console.log(`Confidence: ${block.Confidence}`)

    // Draw box around detected text using polygons
    ctx.strokeStyle = 'rgba(0,0,0,0.5)';
    ctx.beginPath();
    block.Geometry.Polygon.forEach(({X, Y}) =>
    ctx.lineTo(width * X - 10, height * Y - 10)
    );
    ctx.closePath();
    ctx.stroke();
    console.log("-----")
  })

  // render image
  var buffer = canvas.toBuffer("image/png");
  image.draw(images(buffer), 10, 10)
  image.save("output-image.jpg");

} catch (err){
console.error(err);}

}

main()
```

4. Run the example. The Python and Java examples display the document image. A black box surrounds each line of detected text. A green vertical line is the start of a detected word. A red vertical line is the end of a detected word. The AWS CLI example displays only the JSON output for the `DetectDocumentText` operation.

Analyzing Document Text with Amazon Textract

To analyze text in a document, you use the AnalyzeDocument (p. 187) operation, and pass a document file as input. `AnalyzeDocument` returns a JSON structure that contains the analyzed text. For more information, see Analyzing Documents (p. 4).

You can provide an input document as an image byte array (base64-encoded image bytes), or as an Amazon S3 object. In this procedure, you upload an image file to your S3 bucket and specify the file name.

To analyze text in a document (API)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonTextractFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 29).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 29).
2. Upload an image that contains a document to your S3 bucket.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service User Guide*.
3. Use the following examples to call the `AnalyzeDocument` operation.

   Java

   The following example code displays the document and boxes around detected items.

   In the function `main`, replace the values of `bucket` and `document` with the names of the Amazon S3 bucket and document image that you used in step 2.

```
//Loads document from S3 bucket. Displays the document and polygon around detected
 lines of text.
package com.amazonaws.samples;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3ObjectInputStream;
import com.amazonaws.services.textract.AmazonTextract;
import com.amazonaws.services.textract.AmazonTextractClientBuilder;
import com.amazonaws.services.textract.model.AnalyzeDocumentRequest;
import com.amazonaws.services.textract.model.AnalyzeDocumentResult;
import com.amazonaws.services.textract.model.Block;
import com.amazonaws.services.textract.model.BoundingBox;
import com.amazonaws.services.textract.model.Document;
import com.amazonaws.services.textract.model.S3Object;
import com.amazonaws.services.textract.model.Point;
import com.amazonaws.services.textract.model.Relationship;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;

public class AnalyzeDocument extends JPanel {

    private static final long serialVersionUID = 1L;
```

```
   BufferedImage image;

   AnalyzeDocumentResult result;

   public AnalyzeDocument(AnalyzeDocumentResult documentResult, BufferedImage
bufImage) throws Exception {
       super();

       result = documentResult; // Results of text detection.
       image = bufImage; // The image containing the document.

   }

   // Draws the image and text bounding box.
   public void paintComponent(Graphics g) {

       int height = image.getHeight(this);
       int width = image.getWidth(this);

       Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

       // Draw the image.
       g2d.drawImage(image, 0, 0, image.getWidth(this), image.getHeight(this),
this);

       // Iterate through blocks and display bounding boxes around everything.

       List<Block> blocks = result.getBlocks();
       for (Block block : blocks) {
           DisplayBlockInfo(block);
           switch(block.getBlockType()) {

           case "KEY_VALUE_SET":
               if (block.getEntityTypes().contains("KEY")){
                   ShowBoundingBox(height, width,
block.getGeometry().getBoundingBox(), g2d, new Color(255,0,0));
               }
               else {  //VALUE
                   ShowBoundingBox(height, width,
block.getGeometry().getBoundingBox(), g2d, new Color(0,255,0));
               }
               break;
           case "TABLE":
               ShowBoundingBox(height, width,
block.getGeometry().getBoundingBox(), g2d, new Color(0,0,255));
               break;
           case "CELL":
               ShowBoundingBox(height, width,
block.getGeometry().getBoundingBox(), g2d, new Color(255,255,0));
               break;
           case "SELECTION_ELEMENT":
               if (block.getSelectionStatus().equals("SELECTED"))
                   ShowSelectedElement(height, width,
block.getGeometry().getBoundingBox(), g2d, new Color(0,0,255));
               break;
           default:
               //PAGE, LINE & WORD
               //ShowBoundingBox(height, width,
block.getGeometry().getBoundingBox(), g2d, new Color(200,200,0));
           }
       }

        // uncomment to show polygon around all blocks
        //ShowPolygon(height,width,block.getGeometry().getPolygon(),g2d);
   }

   // Show bounding box at supplied location.
   private void ShowBoundingBox(int imageHeight, int imageWidth, BoundingBox box,
Graphics2D g2d, Color color) {

       float left = imageWidth * box.getLeft();
       float top = imageHeight * box.getTop();

       // Display bounding box.
       g2d.setColor(color);
       g2d.drawRect(Math.round(left), Math.round(top),
               Math.round(imageWidth * box.getWidth()), Math.round(imageHeight *
box.getHeight()));

   }
   private void ShowSelectedElement(int imageHeight, int imageWidth, BoundingBox
box, Graphics2D g2d, Color color) {

       float left = imageWidth * box.getLeft();
       float top = imageHeight * box.getTop();

       // Display bounding box.
       g2d.setColor(color);
       g2d.fillRect(Math.round(left), Math.round(top),
               Math.round(imageWidth * box.getWidth()), Math.round(imageHeight *
box.getHeight()));

   }

   // Shows polygon at supplied location
   private void ShowPolygon(int imageHeight, int imageWidth, List<Point> points,
Graphics2D g2d) {

       g2d.setColor(new Color(0, 0, 0));
       Polygon polygon = new Polygon();

       // Construct polygon and display
       for (Point point : points) {
           polygon.addPoint((Math.round(point.getX() * imageWidth)),
                   Math.round(point.getY() * imageHeight));
       }
       g2d.drawPolygon(polygon);
   }
   //Displays information from a block returned by text detection and text
analysis
   private void DisplayBlockInfo(Block block) {
       System.out.println("Block Id : " + block.getId());
       if (block.getText()!=null)
           System.out.println("    Detected text: " + block.getText());
       System.out.println("    Type: " + block.getBlockType());

       if (block.getBlockType().equals("PAGE") !=true) {
           System.out.println("    Confidence: " +
block.getConfidence().toString());
       }
       if(block.getBlockType().equals("CELL"))
       {
           System.out.println("    Cell information:");
           System.out.println("        Column: " + block.getColumnIndex());
           System.out.println("        Row: " + block.getRowIndex());
           System.out.println("        Column span: " + block.getColumnSpan());
           System.out.println("        Row span: " + block.getRowSpan());

       }
```

```
        System.out.println("    Relationships");
        List<Relationship> relationships=block.getRelationships();
        if(relationships!=null) {
            for (Relationship relationship : relationships) {
                System.out.println("        Type: " + relationship.getType());
                System.out.println("        IDs: " +
 relationship.getIds().toString());
            }
        } else {
            System.out.println("        No related Blocks");
        }

        System.out.println("    Geometry");
        System.out.println("        Bounding Box: " +
 block.getGeometry().getBoundingBox().toString());
        System.out.println("        Polygon: " +
 block.getGeometry().getPolygon().toString());

        List<String> entityTypes = block.getEntityTypes();

        System.out.println("    Entity Types");
        if(entityTypes!=null) {
            for (String entityType : entityTypes) {
                System.out.println("        Entity Type: " + entityType);
            }
        } else {
            System.out.println("        No entity type");
        }

        if(block.getBlockType().equals("SELECTION_ELEMENT")) {
            System.out.print("    Selection element detected: ");
            if (block.getSelectionStatus().equals("SELECTED")){
                System.out.println("Selected");
            }else {
                System.out.println(" Not selected");
            }
        }

        if(block.getPage()!=null)
            System.out.println("    Page: " + block.getPage());
        System.out.println();
    }

    public static void main(String arg[]) throws Exception {

        // The S3 bucket and document
        String document = "";
        String bucket = "";

        AmazonS3 s3client = AmazonS3ClientBuilder.standard()
                .withEndpointConfiguration(
                        new EndpointConfiguration("https://s3.amazonaws.com","us-
east-1"))
                .build();


        // Get the document from S3
        com.amazonaws.services.s3.model.S3Object s3object =
 s3client.getObject(bucket, document);
        S3ObjectInputStream inputStream = s3object.getObjectContent();
        BufferedImage image = ImageIO.read(inputStream);

        // Call AnalyzeDocument
        EndpointConfiguration endpoint = new EndpointConfiguration(
                "https://textract.us-east-1.amazonaws.com", "us-east-1");
        AmazonTextract client = AmazonTextractClientBuilder.standard()
                .withEndpointConfiguration(endpoint).build();


        AnalyzeDocumentRequest request = new AnalyzeDocumentRequest()
                .withFeatureTypes("TABLES","FORMS")
                 .withDocument(new Document().
                        withS3Object(new
 S3Object().withName(document).withBucket(bucket)));


        AnalyzeDocumentResult result = client.analyzeDocument(request);

        // Create frame and panel.
        JFrame frame = new JFrame("RotateImage");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        AnalyzeDocument panel = new AnalyzeDocument(result, image);
        panel.setPreferredSize(new Dimension(image.getWidth(), image.getHeight()));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    }
}
```

AWS CLI

This AWS CLI command displays the JSON output for the `detect-document-text` CLI operation.

Replace the values of `Bucket` and `Name` with the names of the Amazon S3 bucket and document that you used in step 2.

```
aws textract analyze-document \
 --document '{"S3Object":{"Bucket":"bucket","Name":"document"}}' \
 --feature-types '["TABLES","FORMS"]'
```

Python

The following example code displays the document and boxes around detected items.

In the function `main`, replace the values of `bucket` and `document` with the names of the Amazon S3 bucket and document that you used in step 2.

```
#Analyzes text in a document stored in an S3 bucket. Display polygon box around
 text and angled text
import boto3
import io
from io import BytesIO
import sys

import math
from PIL import Image, ImageDraw, ImageFont

def ShowBoundingBox(draw,box,width,height,boxColor):

    left = width * box['Left']
    top = height * box['Top']
    draw.rectangle([left,top, left + (width * box['Width']), top +(height *
 box['Height'])],outline=boxColor)
```

```
def ShowSelectedElement(draw,box,width,height,boxColor):

    left = width * box['Left']
    top = height * box['Top']
    draw.rectangle([left,top, left + (width * box['Width']), top +(height *
 box['Height'])],fill=boxColor)

# Displays information about a block returned by text detection and text analysis
def DisplayBlockInformation(block):
    print('Id: {}'.format(block['Id']))
    if 'Text' in block:
        print('    Detected: ' + block['Text'])
    print('    Type: ' + block['BlockType'])

    if 'Confidence' in block:
        print('    Confidence: ' + "{:.2f}".format(block['Confidence']) + "%")

    if block['BlockType'] == 'CELL':
        print("    Cell information")
        print("        Column:" + str(block['ColumnIndex']))
        print("        Row:" + str(block['RowIndex']))
        print("        Column Span:" + str(block['ColumnSpan']))
        print("        RowSpan:" + str(block['ColumnSpan']))

    if 'Relationships' in block:
        print('    Relationships: {}'.format(block['Relationships']))
    print('    Geometry: ')
    print('        Bounding Box: {}'.format(block['Geometry']['BoundingBox']))
    print('        Polygon: {}'.format(block['Geometry']['Polygon']))

    if block['BlockType'] == "KEY_VALUE_SET":
        print ('    Entity Type: ' + block['EntityTypes'][0])

    if block['BlockType'] == 'SELECTION_ELEMENT':
        print('    Selection element detected: ', end='')

        if block['SelectionStatus'] =='SELECTED':
            print('Selected')
        else:
            print('Not selected')

    if 'Page' in block:
        print('Page: ' + block['Page'])
    print()

def process_text_analysis(bucket, document):

    #Get the document from S3
    s3_connection = boto3.resource('s3')

    s3_object = s3_connection.Object(bucket,document)
    s3_response = s3_object.get()

    stream = io.BytesIO(s3_response['Body'].read())
    image=Image.open(stream)

    # Analyze the document
    client = boto3.client('textract')

    image_binary = stream.getvalue()
    response = client.analyze_document(Document={'Bytes': image_binary},
        FeatureTypes=["TABLES", "FORMS"])

    ### Alternatively, process using S3 object ###
    #response = client.analyze_document(
    #    Document={'S3Object': {'Bucket': bucket, 'Name': document}},
    #    FeatureTypes=["TABLES", "FORMS"])

    ### To use a local file ###
    # with open("pathToFile", 'rb') as img_file:
        ### To display image using PIL ###
    #    image = Image.open()
        ### Read bytes ###
    #    img_bytes = img_file.read()
    #    response = client.analyze_document(Document={'Bytes': img_bytes},
 FeatureTypes=["TABLES", "FORMS"])


    #Get the text blocks
    blocks=response['Blocks']
    width, height =image.size
    draw = ImageDraw.Draw(image)
    print ('Detected Document Text')

    # Create image showing bounding box/polygon the detected lines/text
    for block in blocks:

        DisplayBlockInformation(block)

        draw=ImageDraw.Draw(image)
        if block['BlockType'] == "KEY_VALUE_SET":
            if block['EntityTypes'][0] == "KEY":
                ShowBoundingBox(draw, block['Geometry']
['BoundingBox'],width,height,'red')
            else:
                ShowBoundingBox(draw, block['Geometry']
['BoundingBox'],width,height,'green')

        if block['BlockType'] == 'TABLE':
            ShowBoundingBox(draw, block['Geometry']['BoundingBox'],width,height,
 'blue')

        if block['BlockType'] == 'CELL':
            ShowBoundingBox(draw, block['Geometry']['BoundingBox'],width,height,
 'yellow')
        if block['BlockType'] == 'SELECTION_ELEMENT':
            if block['SelectionStatus'] =='SELECTED':
                ShowSelectedElement(draw, block['Geometry']
['BoundingBox'],width,height, 'blue')

            #uncomment to draw polygon for all Blocks
            #points=[]
            #for polygon in block['Geometry']['Polygon']:
            #    points.append((width * polygon['X'], height * polygon['Y']))
            #draw.polygon((points), outline='blue')

    # Display the image
    image.show()
    return len(blocks)


def main():

    bucket = ''
    document = ''
    block_count=process_text_analysis(bucket,document)
    print("Blocks detected: " + str(block_count))

if __name__ == "__main__":
    main()
```

Node.js

The following example code displays the document and boxes around detected items.

In the code below, replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and document that you used in step 2. Replace the value of `region` with the region associated with your account.

```
// Import required AWS SDK clients and commands for Node.js
import { AnalyzeDocumentCommand } from  "@aws-sdk/client-textract";
import  { TextractClient } from "@aws-sdk/client-textract";

// Set the AWS Region.
const REGION = "region"; //e.g. "us-east-1"
// Create SNS service object.
const textractClient = new TextractClient({ region: REGION });

const bucket = 'buckets'
const photo = 'photo'

// Set params
const params = {
    Document: {
      S3Object: {
        Bucket: bucket,
        Name: photo
      },
    },
    FeatureTypes: ['TABLES', 'FORMS'],
  }

const displayBlockInfo = async (response) => {
    try {
        response.Blocks.forEach(block => {
            console.log(`ID: ${block.Id}`)
            console.log(`Block Type: ${block.BlockType}`)
            if ("Text" in block && block.Text !== undefined){
                console.log(`Text: ${block.Text}`)
            }
            else{}
            if ("Confidence" in block && block.Confidence !== undefined){
                console.log(`Confidence: ${block.Confidence}`)
            }
            else{}
            if (block.BlockType == 'CELL'){
                console.log("Cell info:")
                console.log(`   Column Index - ${block.ColumnIndex}`)
                console.log(`   Row - ${block.RowIndex}`)
                console.log(`   Column Span - ${block.ColumnSpan}`)
                console.log(`   Row Span - ${block.RowSpan}`)
            }
            if ("Relationships" in block && block.Relationships !== undefined){
                console.log(block.Relationships)
                console.log("Geometry:")
                console.log(`   Bounding Box -
 ${JSON.stringify(block.Geometry.BoundingBox)}`)
                console.log(`   Polygon -
 ${JSON.stringify(block.Geometry.Polygon)}`)
            }
            console.log("-----")
        });
      } catch (err) {
        console.log("Error", err);
        }
}

const analyze_document_text = async () => {
    try {
        const analyzeDoc = new AnalyzeDocumentCommand(params);
        const response = await textractClient.send(analyzeDoc);
        //console.log(response)
        displayBlockInfo(response)
        return response; // For unit tests.
      } catch (err) {
        console.log("Error", err);
      }
}

analyze_document_text()
```

4. Run the example. The Python and Java examples display the document image with the following colored bounding boxes:

   - Red – KEY Block objects
   - Green – VALUE Block objects
   - Blue – TABLE Block objects
   - Yellow – CELL Block objects

   Selection elements that are selected are filled with blue.

   The AWS CLI example displays only the JSON output for the `AnalyzeDocument` operation.

Analyzing Invoices and Receipts with Amazon Textract

To analyze invoice and receipt documents, you use the AnalyzeExpense API, and pass a document file as input. `AnalyzeExpense` is a synchronous operation that returns a JSON structure that contains the analyzed text. For more information, see Analyzing Invoices and Receipts (p. 5).

You can provide an input document as an image byte array (base64-encoded image bytes), or as an Amazon S3 object. In this procedure, you upload an image file to your S3 bucket and specify the file name.

To analyze an invoice or receipt (API)

1. If you haven't already:

   a. Create or update an IAM user with `AmazonTextractFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 29).

   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 29).

2. Upload an image that contains a document to your S3 bucket.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service User Guide*.

3. Use the following examples to call the `AnalyzeExpense` operation.

CLI

```
  aws textract analyze-expense --document '{"S3Object": {"Bucket": "bucket
 name","Name":
"object name"}}'
```

Python

```
import boto3
import io
from PIL import Image, ImageDraw

def draw_bounding_box(key, val, width, height, draw):
    # If a key is Geometry, draw the bounding box info in it
    if "Geometry" in key:
        # Draw bounding box information
        box = val["BoundingBox"]
        left = width * box['Left']
        top = height * box['Top']
        draw.rectangle([left, top, left + (width * box['Width']), top + (height *
 box['Height'])],
                       outline='black')

# Takes a field as an argument and prints out the detected labels and values
def print_labels_and_values(field):
    # Only if labels are detected and returned
    if "LabelDetection" in field:
        print("Summary Label Detection - Confidence: {}".format(
            str(field.get("LabelDetection")["Confidence"])) + ", "
              + "Summary Values: {}".format(str(field.get("LabelDetection")
["Text"])))
        print(field.get("LabelDetection")["Geometry"])
    else:
        print("Label Detection - No labels returned.")
    if "ValueDetection" in field:
        print("Summary Value Detection - Confidence: {}".format(
            str(field.get("ValueDetection")["Confidence"])) + ", "
              + "Summary Values: {}".format(str(field.get("ValueDetection")
["Text"])))
        print(field.get("ValueDetection")["Geometry"])
    else:
        print("Value Detection - No values returned")

def process_text_detection(bucket, document):
    # Get the document from S3
    s3_connection = boto3.resource('s3')
    s3_object = s3_connection.Object(bucket, document)
    s3_response = s3_object.get()

    # opening binary stream using an in-memory bytes buffer
    stream = io.BytesIO(s3_response['Body'].read())

    # loading stream into image
    image = Image.open(stream)

    # Detect text in the document
    client = boto3.client('textract', region_name="us-east-1")

    # process using S3 object
    response = client.analyze_expense(
        Document={'S3Object': {'Bucket': bucket, 'Name': document}})
    # Set width and height to display image and draw bounding boxes
    # Create drawing object
    width, height = image.size
    draw = ImageDraw.Draw(image)

    for expense_doc in response["ExpenseDocuments"]:
        for line_item_group in expense_doc["LineItemGroups"]:
            for line_items in line_item_group["LineItems"]:
                for expense_fields in line_items["LineItemExpenseFields"]:
                    print_labels_and_values(expense_fields)
                    print()

        print("Summary:")
        for summary_field in expense_doc["SummaryFields"]:
            print_labels_and_values(summary_field)
            print()

        #For draw bounding boxes
        for line_item_group in expense_doc["LineItemGroups"]:
            for line_items in line_item_group["LineItems"]:
                for expense_fields in line_items["LineItemExpenseFields"]:
                    for key, val in expense_fields["ValueDetection"].items():
                        if "Geometry" in key:
                            draw_bounding_box(key, val, width, height, draw)

        for label in expense_doc["SummaryFields"]:
            if "LabelDetection" in label:
                for key, val in label["LabelDetection"].items():
                    draw_bounding_box(key, val, width, height, draw)

    # Display the image
    image.show()

def main():
    bucket = 'Bucket-Name'
    document = 'Document-Name'
    process_text_detection(bucket, document)

if __name__ == "__main__":
    main()
```

Java

```
package com.amazonaws.samples;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import javax.imageio.ImageIO;
import javax.swing.*;
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.async.AsyncResponseTransformer;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.*;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.textract.TextractClient;
import software.amazon.awssdk.services.textract.model.AnalyzeExpenseRequest;
import software.amazon.awssdk.services.textract.model.AnalyzeExpenseResponse;
import software.amazon.awssdk.services.textract.model.BoundingBox;
import software.amazon.awssdk.services.textract.model.Document;
import software.amazon.awssdk.services.textract.model.ExpenseDocument;
import software.amazon.awssdk.services.textract.model.ExpenseField;
import software.amazon.awssdk.services.textract.model.LineItemFields;
import software.amazon.awssdk.services.textract.model.LineItemGroup;
import software.amazon.awssdk.services.textract.model.S3Object;
import software.amazon.awssdk.services.textract.model.Point;

/**
 *
 * Demo code to parse Textract AnalyzeExpense API
 *
 */
public class TextractAnalyzeExpenseSample extends JPanel {

 private static final long serialVersionUID = 1L;

 BufferedImage image;
 static AnalyzeExpenseResponse result;

 public TextractAnalyzeExpenseSample(AnalyzeExpenseResponse documentResult,
 BufferedImage bufImage) throws Exception {
  super();

  result = documentResult; // Results of analyzeexpense summaryfields and
 lineitemgroups detection.
  image = bufImage; // The image containing the document.

 }

 // Draws the image and text bounding box.
 public void paintComponent(Graphics g) {

  Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

  // Draw the image.
  g2d.drawImage(image, 0, 0, image.getWidth(this), image.getHeight(this), this);

  // Iterate through summaryfields and lineitemgroups and display boundedboxes
 around lines of detected label and value.
  List<ExpenseDocument> expenseDocuments = result.expenseDocuments();
  for (ExpenseDocument expenseDocument : expenseDocuments) {

   if (expenseDocument.hasSummaryFields()) {
    DisplayAnalyzeExpenseSummaryInfo(expenseDocument);
    List<ExpenseField> summaryfields = expenseDocument.summaryFields();
    for (ExpenseField summaryfield : summaryfields) {

     if (summaryfield.valueDetection() != null) {
      ShowBoundingBox(image.getHeight(this), image.getWidth(this),
        summaryfield.valueDetection().geometry().boundingBox(), g2d, new Color(0,
 0, 0));
     }

     if (summaryfield.labelDetection() != null) {

      ShowBoundingBox(image.getHeight(this), image.getWidth(this),
        summaryfield.labelDetection().geometry().boundingBox(), g2d, new Color(0,
 0, 0));

     }
```

```
       }

     }

     if (expenseDocument.hasLineItemGroups()) {
      DisplayAnalyzeExpenseLineItemGroupsInfo(expenseDocument);

      List<LineItemGroup> lineitemgroups = expenseDocument.lineItemGroups();

      for (LineItemGroup lineitemgroup : lineitemgroups) {

       if (lineitemgroup.hasLineItems()) {

        List<LineItemFields> lineItems = lineitemgroup.lineItems();
        for (LineItemFields lineitemfield : lineItems) {

         if (lineitemfield.hasLineItemExpenseFields()) {

          List<ExpenseField> expensefields = lineitemfield.lineItemExpenseFields();
          for (ExpenseField expensefield : expensefields) {

           if (expensefield.valueDetection() != null) {
            ShowBoundingBox(image.getHeight(this), image.getWidth(this),
              expensefield.valueDetection().geometry().boundingBox(), g2d,
              new Color(0, 0, 0));
           }

           if (expensefield.labelDetection() != null) {
            ShowBoundingBox(image.getHeight(this), image.getWidth(this),
              expensefield.labelDetection().geometry().boundingBox(), g2d,
              new Color(0, 0, 0));
           }

          }
         }

        }

       }

      }

     }
    }

   }

   // Show bounding box at supplied location.
   private void ShowBoundingBox(float imageHeight, float imageWidth, BoundingBox box,
   Graphics2D g2d, Color color) {

    float left = imageWidth * box.left();
    float top = imageHeight * box.top();

    // Display bounding box.
    g2d.setColor(color);
    g2d.drawRect(Math.round(left), Math.round(top), Math.round(imageWidth *
   box.width()),
      Math.round(imageHeight * box.height()));

   }

   private void ShowSelectedElement(float imageHeight, float imageWidth, BoundingBox
   box, Graphics2D g2d,
     Color color) {
```

```
 float left = (float) imageWidth * (float) box.left();
 float top = (float) imageHeight * (float) box.top();
 System.out.println(left);
 System.out.println(top);

 // Display bounding box.
 g2d.setColor(color);
 g2d.fillRect(Math.round(left), Math.round(top), Math.round(imageWidth *
box.width()),
   Math.round(imageHeight * box.height()));

}

// Shows polygon at supplied location
private void ShowPolygon(int imageHeight, int imageWidth, List<Point> points,
Graphics2D g2d) {

 g2d.setColor(new Color(0, 0, 0));
 Polygon polygon = new Polygon();

 // Construct polygon and display
 for (Point point : points) {
  polygon.addPoint((Math.round(point.x() * imageWidth)), Math.round(point.y() *
imageHeight));
 }
 g2d.drawPolygon(polygon);
}

private void DisplayAnalyzeExpenseSummaryInfo(ExpenseDocument expensedocument) {
 System.out.println(" ExpenseId : " + expensedocument.expenseIndex());
 System.out.println("    Expense Summary information:");
 if (expensedocument.hasSummaryFields()) {

  List<ExpenseField> summaryfields = expensedocument.summaryFields();

  for (ExpenseField summaryfield : summaryfields) {

   System.out.println("    Page: " + summaryfield.pageNumber());
   if (summaryfield.type() != null) {

    System.out.println("    Expense Summary Field Type:" +
summaryfield.type().text());

   }
   if (summaryfield.labelDetection() != null) {

    System.out.println("    Expense Summary Field Label:" +
summaryfield.labelDetection().text());
    System.out.println("    Geometry");
    System.out.println("        Bounding Box: "
      + summaryfield.labelDetection().geometry().boundingBox().toString());
    System.out.println(
      "        Polygon: " +
summaryfield.labelDetection().geometry().polygon().toString());

   }
   if (summaryfield.valueDetection() != null) {
    System.out.println("    Expense Summary Field Value:" +
summaryfield.valueDetection().text());
    System.out.println("    Geometry");
    System.out.println("        Bounding Box: "
      + summaryfield.valueDetection().geometry().boundingBox().toString());
    System.out.println(
      "        Polygon: " +
summaryfield.valueDetection().geometry().polygon().toString());
```

```
    }

   }

  }

 }

 private void DisplayAnalyzeExpenseLineItemGroupsInfo(ExpenseDocument
 expensedocument) {

  System.out.println(" ExpenseId : " + expensedocument.expenseIndex());
  System.out.println("    Expense LineItemGroups information:");

  if (expensedocument.hasLineItemGroups()) {

   List<LineItemGroup> lineitemgroups = expensedocument.lineItemGroups();

   for (LineItemGroup lineitemgroup : lineitemgroups) {

    System.out.println("    Expense LineItemGroupsIndexID :" +
 lineitemgroup.lineItemGroupIndex());

    if (lineitemgroup.hasLineItems()) {

     List<LineItemFields> lineItems = lineitemgroup.lineItems();

     for (LineItemFields lineitemfield : lineItems) {

      if (lineitemfield.hasLineItemExpenseFields()) {

       List<ExpenseField> expensefields = lineitemfield.lineItemExpenseFields();
       for (ExpenseField expensefield : expensefields) {

        if (expensefield.type() != null) {
         System.out.println("    Expense LineItem Field Type:" +
 expensefield.type().text());

        }

        if (expensefield.valueDetection() != null) {
         System.out.println(
           "    Expense Summary Field Value:" +
 expensefield.valueDetection().text());
         System.out.println("    Geometry");
         System.out.println("        Bounding Box: "
           + expensefield.valueDetection().geometry().boundingBox().toString());
         System.out.println("        Polygon: "
           + expensefield.valueDetection().geometry().polygon().toString());

        }

        if (expensefield.labelDetection() != null) {
         System.out.println(
           "    Expense LineItem Field Label:" +
 expensefield.labelDetection().text());
         System.out.println("    Geometry");
         System.out.println("        Bounding Box: "
           + expensefield.labelDetection().geometry().boundingBox().toString());
         System.out.println("        Polygon: "
           + expensefield.labelDetection().geometry().polygon().toString());
        }

       }
      }
     }

    }
   }
  }

 }

 public static void main(String arg[]) throws Exception {

  // Creates a default async client with credentials and AWS Region loaded from
  // the
  // environment


  S3AsyncClient client = S3AsyncClient.builder().region(Region.US_EAST_1).build();

  System.out.println("Creating the S3 Client");

  // Start the call to Amazon S3, not blocking to wait for the result
  CompletableFuture<ResponseBytes<GetObjectResponse>> responseFuture =
 client.getObject(
    GetObjectRequest.builder().bucket("textractanalyzeexpense").key("input/sample-
receipt.jpg").build(),
    AsyncResponseTransformer.toBytes());

  System.out.println("Successfully read the object");

  // When future is complete (either successfully or in error), handle the
  // response
  CompletableFuture<ResponseBytes<GetObjectResponse>> operationCompleteFuture =
 responseFuture
    .whenComplete((getObjectResponse, exception) -> {
     if (getObjectResponse != null) {
      // At this point, the file my-file.out has been created with the data
      // from S3; let's just print the object version
      // Convert this into Async call and remove the below block from here and put
 it
      // outside


      TextractClient textractclient =
 TextractClient.builder().region(Region.US_EAST_1).build();

      AnalyzeExpenseRequest request = AnalyzeExpenseRequest.builder()
        .document(
          Document.builder().s3Object(S3Object.builder().name("YOURObjectName")
            .bucket("YOURBucket").build()).build())
        .build();

      AnalyzeExpenseResponse result = textractclient.analyzeExpense(request);

      System.out.print(result.toString());

      ByteArrayInputStream bais = new
 ByteArrayInputStream(getObjectResponse.asByteArray());
      try {
       BufferedImage image = ImageIO.read(bais);
       System.out.println("Successfully read the image");
       JFrame frame = new JFrame("Expense Image");
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       TextractAnalyzeExpense panel = new TextractAnalyzeExpense(result, image);
       panel.setPreferredSize(new Dimension(image.getWidth(), image.getHeight()));
       frame.setContentPane(panel);
       frame.pack();
       frame.setVisible(true);
      } catch (IOException e) {
       throw new RuntimeException(e);
      } catch (Exception e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
      }
     } else {
      // Handle the error
      exception.printStackTrace();
     }
    });

  // We could do other work while waiting for the AWS call to complete in
  // the background, but we'll just wait for "whenComplete" to finish instead
  operationCompleteFuture.join();

 }
}
```

Node.Js

```
                    // Import required AWS SDK clients and commands for Node.js
import { AnalyzeExpenseCommand } from  "@aws-sdk/client-textract";
import  { TextractClient } from "@aws-sdk/client-textract";

// Set the AWS Region.
const REGION = "region"; //e.g. "us-east-1"
// Create SNS service object.
const textractClient = new TextractClient({ region: REGION });

const bucket = 'bucket'
const photo = 'photo'

// Set params
const params = {
    Document: {
      S3Object: {
        Bucket: bucket,
        Name: photo
      },
    },
  }

const process_text_detection = async () => {
    try {
        const aExpense = new AnalyzeExpenseCommand(params);
        const response = await textractClient.send(aExpense);
        //console.log(response)
        response.ExpenseDocuments.forEach(doc => {
            doc.LineItemGroups.forEach(items => {
                items.LineItems.forEach(fields => {
                    fields.LineItemExpenseFields.forEach(expenseFields =>{
                        console.log(expenseFields)
                    })
                }
                )}
                )
            }
        )
        return response; // For unit tests.
      } catch (err) {
        console.log("Error", err);
      }
}

process_text_detection()
```

4. This will provide you with the JSON output for the `AnalyzeExpense` operation.

Processing Documents with Asynchronous Operations

Amazon Textract can detect and analyze text in multipage documents that are in PDF or TIFF format. This includes invoices and receipts. Multipage document processing is an asynchronous operation. Asynchronous processing of documents is useful for processing large, multipage documents. For example, a PDF file with over 1,000 pages takes a while to process. Processing the PDF file asynchronously allows your application to complete other tasks while it waits for the process to complete.

This section covers how you can use Amazon Textract to asynchronously detect and analyze text on a multipage or single-page document. Multipage documents must be in PDF or TIFF format. Single-page documents processed with asynchronous operations can be in JPEG, PNG, TIFF or PDF format.

You can use Amazon Textract asynchronous operations for the following purposes:

- Text detection – You can detect lines and words on a multipage document. The asynchronous operations are StartDocumentTextDetection (p. 222) and GetDocumentTextDetection (p. 206). For more information, see Detecting Text (p. 3).
- Text analysis – You can identify relationships between detected text on a multipage document. The asynchronous operations are StartDocumentAnalysis (p. 217) and GetDocumentAnalysis (p. 201). For more information, see Analyzing Documents (p. 4).
- Expense analysis – You can identify data relationships on multipage invoices and receipts. Amazon Textract treats each invoice or a receipt page of a multi-page document as an individual receipt or an invoice. It does not retain the context from one page to another of a multi-page document. The asynchronous operations are StartExpenseAnalysis (p. 226) and GetExpenseAnalysis (p. 211). For more information, see Analyzing Invoices and Receipts (p. 5).

Topics

Calling Amazon Textract Asynchronous Operations

Amazon Textract provides an asynchronous API that you can use to process multipage documents in PDF or TIFF format. You can also use asynchronous operations to process single-page documents that are in JPEG, PNG, TIFF, or PDF format.

The information in this topic uses text detection operations to show how to use Amazon Textract asynchronous operations. The same approach works with the text analysis operations of the section called " StartDocumentAnalysis " (p. 217) and the section called " GetDocumentAnalysis " (p. 201). It also works the same with the section called " StartExpenseAnalysis " (p. 226) and the section called " GetExpenseAnalysis " (p. 211).

For an example, see Detecting or Analyzing Text in a Multipage Document (p. 119).

Amazon Textract asynchronously processes a document stored in an Amazon S3 bucket. You start processing by calling a `Start` operation, such as StartDocumentTextDetection (p. 222). The completion status of the request is published to an Amazon Simple Notification Service (Amazon SNS) topic. To get the completion status from the Amazon SNS topic, you can use an Amazon Simple Queue Service (Amazon SQS) queue or an AWS Lambda function. After you have the completion status, you call a `Get` operation, such as GetDocumentTextDetection (p. 206), to get the results of the request.

Results of asynchronous calls are encrypted and stored for 7 days in a Amazon Textract owned bucket by default, unless you specify an Amazon S3 bucket using an operation's `OutputConfig` argument.

The following table shows the corresponding Start and Get operations for the different types of asynchronous processing supported by Amazon Textract:

Start/Get API Operations for Amazon Textract Asynchronous Operations

Processing Type	Start API	Get API
Text Detection	StartDocumentTextDetection	GetDocumentTextDetection
Text Analysis	StartDocumentAnalysis	GetDocumentAnalysis
Expense Analysis	StartExpenseAnalysis	GetExpenseAnalysis

For an example that uses AWS Lambda functions, see Large scale document processing with Amazon Textract.

The following diagram shows the process for detecting document text in a document image stored in an Amazon S3 bucket. In the diagram, an Amazon SQS queue gets the completion status from the Amazon SNS topic.

The process displayed by the preceeding diagram is the same for analyzing text and invoices/receipts. You start analyzing text by calling the section called “ StartDocumentAnalysis ” (p. 217) and start analyzing invoices/receipts by calling the section called “ StartExpenseAnalysis ” (p. 226) You get the results by calling the section called “ GetDocumentAnalysis ” (p. 201) or the section called “ GetExpenseAnalysis ” (p. 211) respectively.

Starting Text Detection

You start an Amazon Textract text detection request by calling StartDocumentTextDetection (p. 222). The following is an example of a JSON request that's passed by `StartDocumentTextDetection`.

```
{
    "DocumentLocation": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "image.pdf"
        }
    },
    "ClientRequestToken": "DocumentDetectionToken",
    "NotificationChannel": {
        "SNSTopicArn": "arn:aws:sns:us-east-1:nnnnnnnnnn:topic",
        "RoleArn": "arn:aws:iam::nnnnnnnnnn:role/roleTopic"
    },
    "JobTag": "Receipt"
}
```

The input parameter `DocumentLocation` provides the document file name and the Amazon S3 bucket to retrieve it from. `NotificationChannel` contains the Amazon Resource Name (ARN) of the Amazon SNS topic that Amazon Textract notifies when the text detection request finishes. The Amazon SNS topic must be in the same AWS Region as the Amazon Textract endpoint that you're calling.

`NotificationChannel` also contains the ARN for a role that allows Amazon Textract to publish to the Amazon SNS topic. You give Amazon Textract publishing permissions to your Amazon SNS topics by creating an IAM service role. For more information, see Configuring Amazon Textract for Asynchronous Operations (p. 118).

You can also specify an optional input parameter, `JobTag`, that enables you to identify the job, or groups of jobs, in the completion status that's published to the Amazon SNS topic. For example, you can use `JobTag` to identify the type of document being processed, such as a tax form or receipt.

To prevent accidental duplication of analysis jobs, you can optionally provide an idempotent token, `ClientRequestToken`. If you supply a value for `ClientRequestToken`, the `Start` operation returns the same `JobId` for multiple identical calls to the `Start` operation, such as `StartDocumentTextDetection`. A `ClientRequestToken` token has a lifetime of 7 days. After 7 days, you can reuse it. If you reuse the token during the token lifetime, the following happens:

- If you reuse the token with same `Start` operation and the same input parameters, the same `JobId` is returned. The job isn't performed again and Amazon Textract doesn't send a completion status to the registered Amazon SNS topic.
- If you reuse the token with the same `Start` operation and a minor input parameter change, you get an `idempotentparametermismatchexception` (HTTP status code: 400) exception raised.
- If you reuse the token with a different `Start` operation, the operation succeeds.

Another optional parameter available is `OutputConfig`, which lets you adjust where your output will be placed. By default, Amazon Textract will store the results internally, and can only be accessed by the Get API operations. With `OutputConfig` enabled, you can set the name of the bucket the output will be sent to, and the file prefix of the results, where you can download your results. Additionally, you can set the `KMSKeyID` parameter to a customer managed key to encrypt your output. Without this parameter set Amazon Textract will encrypt server-side using the AWS managed key for Amazon S3

> **Note**
> Before using this parameter, ensure you have the PutObject permission for the output bucket. Additionally, ensure you have the Decrypt, ReEncrypt, GenerateDataKey, and DescribeKey permissions for the AWS KMS key if you decide to use it.

The response to the `StartDocumentTextDetection` operation is a job identifier (`JobId`). Use `JobId` to track requests and get the analysis results after Amazon Textract has published the completion status to the Amazon SNS topic. The following is an example:

```
{"JobId":"270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3"}
```

If you start too many jobs concurrently, calls to `StartDocumentTextDetection` raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Textract service limit.

If you find that LimitExceededException exceptions are raised with bursts of activity, consider using an Amazon SQS queue to manage incoming requests. Contact AWS Support if you find that your average number of concurrent requests can't be managed by an Amazon SQS queue and you're still receiving `LimitExceededException` exceptions.

Getting the Completion Status of an Amazon Textract Analysis Request

Amazon Textract sends an analysis completion notification to the registered Amazon SNS topic. The notification includes the job identifier and the completion status of the operation in a JSON string. A successful text detection request has a `SUCCEEDED` status. For example, the following result shows the successful processing of a text detection job.

```
{
    "JobId": "642492aea78a86a40665555dc375ee97bc963f342b29cd05030f19bd8fd1bc5f",
    "Status": "SUCCEEDED",
    "API": "StartDocumentTextDetection",
    "JobTag": "Receipt",
    "Timestamp": 1543599965969,
    "DocumentLocation": {
        "S3ObjectName": "document",
        "S3Bucket": "bucket"
    }
}
```

For more information, see Amazon Textract Results Notification (p. 132).

To get the status information published to the Amazon SNS topic by Amazon Textract, use one of the following options:

- **AWS Lambda** – You can subscribe an AWS Lambda function that you write to an Amazon SNS topic. The function is called when Amazon Textract notifies the Amazon SNS topic that the request has completed. Use a Lambda function if you want server-side code to process the results of a text detection request. For example, you might want to use server-side code to annotate the image or create a report on the detected text before returning the information to a client application.
- **Amazon SQS** – You can subscribe an Amazon SQS queue to an Amazon SNS topic. You then poll the Amazon SQS queue to retrieve the completion status published by Amazon Textract when a text detection request completes. For more information, see Detecting or Analyzing Text in a Multipage Document (p. 119). Use an Amazon SQS queue if you want to call Amazon Textract operations only from a client application.

Important
We don't recommend getting the request completion status by repeatedly calling the Amazon Textract `Get` operation. This is because Amazon Textract throttles the `Get` operation if too many requests are made. If you're processing multiple documents at the same time, it's simpler and more efficient to monitor one SQS queue for the completion notification than to poll Amazon Textract for the status of each job individually.

Getting Amazon Textract Text Detection Results

To get the results of a text detection request, first ensure that the completion status that's retrieved from the Amazon SNS topic is `SUCCEEDED`. Then call `GetDocumentTextDetection`, which passes the `JobId` value that's returned from `StartDocumentTextDetection`. The request JSON is similar to the following example:

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3",
    "MaxResults": 10,
    "SortBy": "TIMESTAMP"
}
```

`JobId` is the identifier for the text detection operation. Because text detection can generate large amounts of data, use `MaxResults` to specify the maximum number of results to return in a single `Get`operation. The default value for `MaxResults` is 1,000. If you specify a value greater than 1,000, only 1,000 results are returned. If the operation doesn't return all of the results, a pagination token for the next page is returned. To get the next page of results, specify the token in the `NextToken` parameter.

Note
Amazon Textract retains the results of asynchronous operations for 7 days. You can't retrieve the results after this time.

The `GetDocumentTextDetection` operation response JSON is similar to the following. The total number of pages that are detected is returned in `DocumentMetadata`. The detected text is returned in the `Blocks` array. For information about `Block` objects, see Text Detection and Document Analysis Response Objects (p. 8).

```
{
    "DocumentMetadata": {
        "Pages": 1
    },
    "JobStatus": "SUCCEEDED",
    "Blocks": [
        {
            "BlockType": "PAGE",
            "Geometry": {
                "BoundingBox": {
                    "Width": 1.0,
                    "Height": 1.0,
                    "Left": 0.0,
                    "Top": 0.0
                },
                "Polygon": [
                    {
                        "X": 0.0,
                        "Y": 0.0
                    },
                    {
                        "X": 1.0,
                        "Y": 0.0
                    },
                    {
                        "X": 1.0,
                        "Y": 1.0
                    },
                    {
                        "X": 0.0,
                        "Y": 1.0
                    }
                ]
            },
            "Id": "64533157-c47e-401a-930e-7ca1bb3ac3fa",
            "Relationships": [
                {
                    "Type": "CHILD",
                    "Ids": [
                        "4297834d-dcb1-413b-8908-3b96866ebbb5",
                        "1d85ba24-2877-4d09-b8b2-393833d769e9",
                        "193e9c47-fd87-475a-ba09-3fda210d8784",
                        "bd8aeb62-961b-4b47-b78a-e4ed9eeecd0f"
                    ]
                }
            ],
            "Page": 1
        },
        {
            "BlockType": "LINE",
            "Confidence": 53.301639556884766,
            "Text": "ellooworio",
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.9999999403953552,
                    "Height": 0.5365243554115295,
                    "Left": 0.0,
                    "Top": 0.46347561478614807
                },
                "Polygon": [
                    {
                        "X": 0.0,
                        "Y": 0.46347561478614807
                    },
                    {
                        "X": 0.9999999403953552,
                        "Y": 0.46347561478614807
                    },
                    {
                        "X": 0.9999999403953552,
                        "Y": 1.0
                    },
                    {
                        "X": 0.0,
                        "Y": 1.0
                    }
                ]
            },
            "Id": "4297834d-dcb1-413b-8908-3b96866ebbb5",
            "Relationships": [
                {
                    "Type": "CHILD",
                    "Ids": [
                        "170c3eb9-5155-4bec-8c44-173bba537e70"
                    ]
                }
            ],
            "Page": 1
        },
        {
            "BlockType": "LINE",
            "Confidence": 89.15632629394531,
            "Text": "He llo,",
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33642634749412537,
                    "Height": 0.49159330129623413,
                    "Left": 0.13885067403316498,
                    "Top": 0.17169663310050964
                },
                "Polygon": [
                    {
                        "X": 0.13885067403316498,
                        "Y": 0.17169663310050964
                    },
                    {
                        "X": 0.47527703642845154,
                        "Y": 0.17169663310050964
                    },
                    {
                        "X": 0.47527703642845154,
                        "Y": 0.6632899641990662
                    },
                    {
                        "X": 0.13885067403316498,
                        "Y": 0.6632899641990662
                    }
                ]
            },
            "Id": "1d85ba24-2877-4d09-b8b2-393833d769e9",
            "Relationships": [
                {
                    "Type": "CHILD",
                    "Ids": [
                        "516ae823-3bab-4f9a-9d74-ad7150d128ab",
                        "6bcf4ea8-bbe8-4686-91be-b98dd63bc6a6"
                    ]
                }
            ],
            "Page": 1
        },
        {
            "BlockType": "LINE",
            "Confidence": 82.44834899902344,
            "Text": "worlo",
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33182239532470703,
                    "Height": 0.3766750991344452,
                    "Left": 0.5091826915740967,
                    "Top": 0.23131252825260162
                },
                "Polygon": [
                    {
                        "X": 0.5091826915740967,
                        "Y": 0.23131252825260162
                    },
                    {
                        "X": 0.8410050868988037,
                        "Y": 0.23131252825260162
                    },
                    {
                        "X": 0.8410050868988037,
                        "Y": 0.607987642288208
                    },
                    {
                        "X": 0.5091826915740967,
                        "Y": 0.607987642288208
                    }
                ]
            },
            "Id": "193e9c47-fd87-475a-ba09-3fda210d8784",
            "Relationships": [
                {
                    "Type": "CHILD",
                    "Ids": [
                        "ed135c3b-35dd-4085-8f00-26aedab0125f"
                    ]
                }
            ],
            "Page": 1
        },
        {
            "BlockType": "LINE",
            "Confidence": 88.50325775146484,
            "Text": "world",
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.35004907846450806,
                    "Height": 0.19635874032974243,
                    "Left": 0.527581512928009,
                    "Top": 0.30100569128990173
                },
                "Polygon": [
                    {
                        "X": 0.527581512928009,
                        "Y": 0.30100569128990173
                    },
                    {
                        "X": 0.8776305913925171,
                        "Y": 0.30100569128990173
                    },
                    {
                        "X": 0.8776305913925171,
                        "Y": 0.49736443161964417
                    },
                    {
                        "X": 0.527581512928009,
                        "Y": 0.49736443161964417
                    }
                ]
            },
            "Id": "bd8aeb62-961b-4b47-b78a-e4ed9eeecd0f",
            "Relationships": [
                {
                    "Type": "CHILD",
                    "Ids": [
                        "9e28834d-798e-4a62-8862-a837dfd895a6"
                    ]
                }
            ],
            "Page": 1
        },
        {
            "BlockType": "WORD",
            "Confidence": 53.301639556884766,
            "Text": "ellooworio",
            "Geometry": {
                "BoundingBox": {
                    "Width": 1.0,
                    "Height": 0.5365243554115295,
                    "Left": 0.0,
                    "Top": 0.46347561478614807
                },
                "Polygon": [
                    {
                        "X": 0.0,
                        "Y": 0.46347561478614807
                    },
                    {
                        "X": 1.0,
                        "Y": 0.46347561478614807
                    },
                    {
                        "X": 1.0,
                        "Y": 1.0
                    },
                    {
                        "X": 0.0,
                        "Y": 1.0
                    }
                ]
            },
            "Id": "170c3eb9-5155-4bec-8c44-173bba537e70",
            "Page": 1
        },
        {
            "BlockType": "WORD",
            "Confidence": 88.46246337890625,
            "Text": "He",
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.15350718796253204,
                    "Height": 0.2995560765266418 5,
                    "Left": 0.13885067403316498,
                    "Top": 0.21856294572353363
                },
                "Polygon": [
                    {
                        "X": 0.13885067403316498,
                        "Y": 0.21856294572353363
                    },
                    {
                        "X": 0.292357861995697,
                        "Y": 0.21856294572353363
                    },
                    {
                        "X": 0.292357861995697,
                        "Y": 0.5181190371513367
                    },
                    {
                        "X": 0.13885067403316498,
                        "Y": 0.5181190371513367
                    }
                ]
            },
            "Id": "516ae823-3bab-4f9a-9d74-ad7150d128ab",
            "Page": 1
        },
        {
            "BlockType": "WORD",
            "Confidence": 89.8501968383789,
            "Text": "llo,",
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.17724157869815826,
                    "Height": 0.49159327149391174,
                    "Left": 0.2980354428291321,
                    "Top": 0.17169663310050964
                },
                "Polygon": [
                    {
                        "X": 0.2980354428291321,
                        "Y": 0.17169663310050964
                    },
                    {
                        "X": 0.47527703642845154,
                        "Y": 0.17169663310050964
                    },
                    {
                        "X": 0.47527703642845154,
                        "Y": 0.6632899045944214
                    },
                    {
                        "X": 0.2980354428291321,
                        "Y": 0.6632899045944214
                    }
                ]
            },
            "Id": "6bcf4ea8-bbe8-4686-91be-b98dd63bc6a6",
            "Page": 1
        },
        {
            "BlockType": "WORD",
            "Confidence": 82.44834899902344,
            "Text": "worlo",
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33182239532470703,
                    "Height": 0.3766750991344452,
                    "Left": 0.5091826915740967,
                    "Top": 0.2313125282525260162
                },
                "Polygon": [
                    {
                        "X": 0.5091826915740967,
                        "Y": 0.23131252825260162
                    },
                    {
                        "X": 0.8410050868988037,
                        "Y": 0.23131252825260162
                    },
                    {
                        "X": 0.8410050868988037,
                        "Y": 0.607987642288208
                    },
                    {
                        "X": 0.5091826915740967,
                        "Y": 0.607987642288208
                    }
                ]
            },
            "Id": "ed135c3b-35dd-4085-8f00-26aedab0125f",
            "Page": 1
        },
        {
            "BlockType": "WORD",
            "Confidence": 88.50325775146484,
            "Text": "world",
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.35004907846450806,
                    "Height": 0.19635874032974243,
                    "Left": 0.527581512928009,
                    "Top": 0.30100569128990173
                },
                "Polygon": [
                    {
                        "X": 0.527581512928009,
                        "Y": 0.30100569128990173
                    },
                    {
                        "X": 0.8776305913925171,
                        "Y": 0.30100569128990173
                    },
                    {
                        "X": 0.8776305913925171,
                        "Y": 0.49736443161964417
                    },
                    {
                        "X": 0.527581512928009,
                        "Y": 0.49736443161964417
                    }
                ]
            },
            "Id": "9e28834d-798e-4a62-8862-a837dfd895a6",
            "Page": 1
        }
    ]
}
```

Configuring Amazon Textract for Asynchronous Operations

The following procedures show you how to configure Amazon Textract to use with an Amazon Simple Notification Service (Amazon SNS) topic and an Amazon Simple Queue Service (Amazon SQS) queue.

Note
If you're using these instructions to set up the Detecting or Analyzing Text in a Multipage Document (p. 119) example, you don't need to do steps 3 – 6. The example includes code to create and configure the Amazon SNS topic and Amazon SQS queue.

To configure Amazon Textract

1. Set up an AWS account to access Amazon Textract. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 28).

   Ensure that the user has at least the following permissions:

   - AmazonTextractFullAccess
   - AmazonS3ReadOnlyAccess
   - AmazonSNSFullAccess
   - AmazonSQSFullAccess
2. Install and configure the required AWS SDK. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 29).
3. Create an Amazon SNS topic. Prepend the topic name with *AmazonTextract*. Note the topic Amazon Resource Name (ARN). Ensure that the topic is in the same Region as the AWS endpoint that you're using with your AWS account.
4. Create an Amazon SQS standard queue by using the Amazon SQS console. Note the queue ARN.
5. Subscribe the queue to the topic you created in step 3.
6. Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue.
7. Create an IAM service role to give Amazon Textract access to your Amazon SNS topics. Note the Amazon Resource Name (ARN) of the service role. For more information, see Giving Amazon Textract Access to Your Amazon SNS Topic (p. 119).
8. Add the following inline policy to the IAM user that you created in step 1.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MySid",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "Service role ARN from step 7"
        }
    ]
}
```

   Give the inline policy a name.
9. You can now run the examples in Detecting or Analyzing Text in a Multipage Document (p. 119).

Giving Amazon Textract Access to Your Amazon SNS Topic

Amazon Textract needs permission to send a message to your Amazon SNS topic when an asynchronous operation is complete. to your Amazon SNS topic. You use an IAM service role to give Amazon Textract access to the Amazon SNS topic.

When you create the Amazon SNS topic, you must prepend the topic name with **`AmazonTextract`**—for example, **`AmazonTextractMyTopicName`**.

1. Sign in to the IAM console (https://console.aws.amazon.com/iam).
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. For **Select type of trusted entity**, choose **AWS service**.
5. For **Choose the service that will use this role**, choose **Textract**.
6. Choose **Next: Permissions**.
7. Verify that the **AmazonTextractServiceRole** policy has been included in the list of attached policies. To display the policy in the list, enter part of the policy name in the **Filter policies**.
8. Choose **Next: Tags**.
9. You don't need to add tags, so choose **Next: Review**.
10. In the **Review** section, for **Role name**, enter a name for the role (for example, `TextractRole`). In **Role description**, update the description for the role, and then choose **Create role**.
11. Choose the new role to open the role's details page.
12. In the **Summary**, copy the **Role ARN** value and save it.
13. Choose **Trust relationships**.
14. Choose **Edit trust relationship**, and ensure that the trust policy looks as follows.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "textract.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

15. Choose **Update Trust Policy**.

Detecting or Analyzing Text in a Multipage Document

This procedure shows you how to detect or analyze text in a multipage document by using Amazon Textract detection operations, a document stored in an Amazon S3 bucket, an Amazon SNS topic, and an Amazon SQS queue. Multipage document processing is an asynchronous operation. For more information, see Calling Amazon Textract Asynchronous Operations (p. 108).

You can choose the type of processing that you want the code to do: text detection, text analysis, or expense analysis.

The processing results are returned in an array of the section called “ Block ” (p. 231) objects, which differ depending on the type of processing you use.

To analyzing or detect text in a multipage documents, you do the following:

1. Create the Amazon SNS topic and the Amazon SQS queue.
2. Subscribe the queue the topic.
3. Give the topic permission to send messages to the queue.
4. Start processing the document. Use the appropriate operation for you chose type of analysis:
   - StartDocumentTextDetection (p. 222) for text detection tasks.
   - StartDocumentAnalysis (p. 217) for text analysis tasks.
   - StartExpenseAnalysis (p. 226) for expense analysis tasks.
5. Get the completion status from the Amazon SQS queue. The example code tracks the job identifier (`JobId`) that's returned by the `Start` operation. It only gets the results for matching job identifiers that are read from the completion status. This is important if other applications are using the same queue and topic. For simplicity, the example deletes jobs that don't match. Consider adding the deleted jobs to an Amazon SQS dead-letter queue for further investigation.
6. Get and display the processing results by calling the appropriate operation for your chosen type of analysis:
   - GetDocumentTextDetection (p. 206) for text detection tasks.
   - GetDocumentAnalysis (p. 201) for text analysis tasks.
   - GetExpenseAnalysis (p. 211) for expense analysis tasks.
7. Delete the Amazon SNS topic and the Amazon SQS queue.

Performing Asynchronous Operations

The example code for this procedure is provided in Java, Python, and the AWS CLI. Before you begin, install the appropriate AWS SDK. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 29).

To detect or analyze text in a multipage document

1. Configure user access to Amazon Textract, and configure Amazon Textract access to Amazon SNS. For more information, see Configuring Amazon Textract for Asynchronous Operations (p. 118). To complete this procedure, you need a multipage document file in PDF format. Skip steps 3 – 6 because the example code creates and configures the Amazon SNS topic and Amazon SQS queue. If completing the CLI example, you don't need to set up an SQS queue.
2. Upload a multipage document file in PDF or TIFF format to your Amazon S3 bucket. (Single-page documents in JPEG, PNG, TIFF, or PDF format can also be processed).

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service User Guide*.
3. Use the following AWS SDK for Java, SDK for Python (Boto3), or AWS CLI code to either detect text or analyze text in a multipage document. In the `main` function:

   - Replace the value of `roleArn` with the IAM role ARN that you saved in Giving Amazon Textract Access to Your Amazon SNS Topic (p. 119).
   - Replace the values of `bucket` and `document` with the bucket and document file name that you specified in step 2.
   - Replace the value of the `type` input parameter of the `ProcessDocument` function with the type of processing that you want to do. Use `ProcessType.DETECTION` to detect text. Use `ProcessType.ANALYSIS` to analyze text.

- For the Python example, replace the value of `region_name` with the region your client is operating in.

For the AWS CLI example, do the following:

- When calling StartDocumentTextDetection (p. 222), replace the value of `bucket-name` with the name of your S3 bucket, and replace `file-name` with the name of the file you specified in step 2. Specify the region of your bucket by replacing `region-name` with the name of your region. Take note that the CLI example does not make use of SQS.
- When calling GetDocumentTextDetection (p. 206) replace `job-id-number` with the `job-id` returned by StartDocumentTextDetection (p. 222). Specify the region of your bucket by replacing `region-name` with the name of your region.

Java

```
package com.amazonaws.samples;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.amazonaws.auth.policy.Condition;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Principal;
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.Statement.Effect;
import com.amazonaws.auth.policy.actions.SQSActions;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.CreateTopicResult;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.QueueAttributeName;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
import com.amazonaws.services.textract.AmazonTextract;
import com.amazonaws.services.textract.AmazonTextractClientBuilder;
import com.amazonaws.services.textract.model.Block;
import com.amazonaws.services.textract.model.DocumentLocation;
import com.amazonaws.services.textract.model.DocumentMetadata;
import com.amazonaws.services.textract.model.GetDocumentAnalysisRequest;
import com.amazonaws.services.textract.model.GetDocumentAnalysisResult;
import com.amazonaws.services.textract.model.GetDocumentTextDetectionRequest;
import com.amazonaws.services.textract.model.GetDocumentTextDetectionResult;
import com.amazonaws.services.textract.model.NotificationChannel;
import com.amazonaws.services.textract.model.Relationship;
import com.amazonaws.services.textract.model.S3Object;
import com.amazonaws.services.textract.model.StartDocumentAnalysisRequest;
import com.amazonaws.services.textract.model.StartDocumentAnalysisResult;
import com.amazonaws.services.textract.model.StartDocumentTextDetectionRequest;
import com.amazonaws.services.textract.model.StartDocumentTextDetectionResult;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;;
public class DocumentProcessor {

    private static String sqsQueueName=null;
    private static String snsTopicName=null;
```

```
   private static String snsTopicArn = null;
   private static String roleArn= null;
   private static String sqsQueueUrl = null;
   private static String sqsQueueArn = null;
   private static String startJobId = null;
   private static String bucket = null;
   private static String document = null;
   private static AmazonSQS sqs=null;
   private static AmazonSNS sns=null;
   private static AmazonTextract textract = null;

   public enum ProcessType {
       DETECTION,ANALYSIS
   }

   public static void main(String[] args) throws Exception {

       String document = "document";
       String bucket = "bucket";
       String roleArn="role";

       sns = AmazonSNSClientBuilder.defaultClient();
       sqs= AmazonSQSClientBuilder.defaultClient();
       textract=AmazonTextractClientBuilder.defaultClient();

       CreateTopicandQueue();
       ProcessDocument(bucket,document,roleArn,ProcessType.DETECTION);
       DeleteTopicandQueue();
       System.out.println("Done!");


   }
   // Creates an SNS topic and SQS queue. The queue is subscribed to the topic.
   static void CreateTopicandQueue()
   {
       //create a new SNS topic
       snsTopicName="AmazonTextractTopic" +
Long.toString(System.currentTimeMillis());
       CreateTopicRequest createTopicRequest = new
CreateTopicRequest(snsTopicName);
       CreateTopicResult createTopicResult = sns.createTopic(createTopicRequest);
       snsTopicArn=createTopicResult.getTopicArn();

       //Create a new SQS Queue
       sqsQueueName="AmazonTextractQueue" +
Long.toString(System.currentTimeMillis());
       final CreateQueueRequest createQueueRequest = new
CreateQueueRequest(sqsQueueName);
       sqsQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
       sqsQueueArn = sqs.getQueueAttributes(sqsQueueUrl,
Arrays.asList("QueueArn")).getAttributes().get("QueueArn");

       //Subscribe SQS queue to SNS topic
       String sqsSubscriptionArn = sns.subscribe(snsTopicArn, "sqs",
sqsQueueArn).getSubscriptionArn();

       // Authorize queue
         Policy policy = new Policy().withStatements(
                 new Statement(Effect.Allow)
                 .withPrincipals(Principal.AllUsers)
                 .withActions(SQSActions.SendMessage)
                 .withResources(new Resource(sqsQueueArn))
                 .withConditions(new
Condition().withType("ArnEquals").withConditionKey("aws:SourceArn").withValues(snsTopicArn))
                 );
         Map queueAttributes = new HashMap();
         queueAttributes.put(QueueAttributeName.Policy.toString(),
policy.toJson());
         sqs.setQueueAttributes(new SetQueueAttributesRequest(sqsQueueUrl,
queueAttributes));


        System.out.println("Topic arn: " + snsTopicArn);
        System.out.println("Queue arn: " + sqsQueueArn);
        System.out.println("Queue url: " + sqsQueueUrl);
        System.out.println("Queue sub arn: " + sqsSubscriptionArn );
     }
   static void DeleteTopicandQueue()
   {
       if (sqs !=null) {
           sqs.deleteQueue(sqsQueueUrl);
           System.out.println("SQS queue deleted");
       }

       if (sns!=null) {
           sns.deleteTopic(snsTopicArn);
           System.out.println("SNS topic deleted");
       }
   }

   //Starts the processing of the input document.
   static void ProcessDocument(String inBucket, String inDocument, String
inRoleArn, ProcessType type) throws Exception
   {
       bucket=inBucket;
       document=inDocument;
       roleArn=inRoleArn;

       switch(type)
       {
           case DETECTION:
               StartDocumentTextDetection(bucket, document);
               System.out.println("Processing type: Detection");
               break;
           case ANALYSIS:
               StartDocumentAnalysis(bucket,document);
               System.out.println("Processing type: Analysis");
               break;
           default:
               System.out.println("Invalid processing type. Choose Detection or
Analysis");
               throw new Exception("Invalid processing type");

       }

       System.out.println("Waiting for job: " + startJobId);
       //Poll queue for messages
       List<Message> messages=null;
       int dotLine=0;
       boolean jobFound=false;

       //loop until the job status is published. Ignore other messages in queue.
       do{
           messages = sqs.receiveMessage(sqsQueueUrl).getMessages();
           if (dotLine++<40){
               System.out.print(".");
           }else{
               System.out.println();
               dotLine=0;
           }
```

```
            if (!messages.isEmpty()) {
                //Loop through messages received.
                for (Message message: messages) {
                    String notification = message.getBody();

                    // Get status and job id from notification.
                    ObjectMapper mapper = new ObjectMapper();
                    JsonNode jsonMessageTree = mapper.readTree(notification);
                    JsonNode messageBodyText = jsonMessageTree.get("Message");
                    ObjectMapper operationResultMapper = new ObjectMapper();
                    JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                    JsonNode operationJobId = jsonResultTree.get("JobId");
                    JsonNode operationStatus = jsonResultTree.get("Status");
                    System.out.println("Job found was " + operationJobId);
                    // Found job. Get the results and display.
                    if(operationJobId.asText().equals(startJobId)){
                        jobFound=true;
                        System.out.println("Job id: " + operationJobId );
                        System.out.println("Status : " +
operationStatus.toString());
                        if (operationStatus.asText().equals("SUCCEEDED")){
                            switch(type)
                            {
                                case DETECTION:
                                    GetDocumentTextDetectionResults();
                                    break;
                                case ANALYSIS:
                                    GetDocumentAnalysisResults();
                                    break;
                                default:
                                    System.out.println("Invalid processing type.
Choose Detection or Analysis");
                                    throw new Exception("Invalid processing type");

                            }
                        }
                        else{
                            System.out.println("Document analysis failed");
                        }

                        sqs.deleteMessage(sqsQueueUrl,message.getReceiptHandle());
                    }

                    else{
                        System.out.println("Job received was not job " +
startJobId);
                        //Delete unknown message. Consider moving message to dead
letter queue
                        sqs.deleteMessage(sqsQueueUrl,message.getReceiptHandle());
                    }
                }
            }
            else {
                Thread.sleep(5000);
            }
        } while (!jobFound);

        System.out.println("Finished processing document");
    }

    private static void StartDocumentTextDetection(String bucket, String document)
throws Exception{

        //Create notification channel
```

```
       NotificationChannel channel= new NotificationChannel()
               .withSNSTopicArn(snsTopicArn)
               .withRoleArn(roleArn);

       StartDocumentTextDetectionRequest req = new
StartDocumentTextDetectionRequest()
               .withDocumentLocation(new DocumentLocation()
                   .withS3Object(new S3Object()
                       .withBucket(bucket)
                       .withName(document)))
               .withJobTag("DetectingText")
               .withNotificationChannel(channel);

       StartDocumentTextDetectionResult startDocumentTextDetectionResult =
textract.startDocumentTextDetection(req);
       startJobId=startDocumentTextDetectionResult.getJobId();
   }

 //Gets the results of processing started by StartDocumentTextDetection
   private static void GetDocumentTextDetectionResults() throws Exception{
       int maxResults=1000;
       String paginationToken=null;
       GetDocumentTextDetectionResult response=null;
       Boolean finished=false;

       while (finished==false)
       {
           GetDocumentTextDetectionRequest documentTextDetectionRequest= new
GetDocumentTextDetectionRequest()
                   .withJobId(startJobId)
                   .withMaxResults(maxResults)
                   .withNextToken(paginationToken);
           response =
textract.getDocumentTextDetection(documentTextDetectionRequest);
           DocumentMetadata documentMetaData=response.getDocumentMetadata();

           System.out.println("Pages: " + documentMetaData.getPages().toString());

           //Show blocks information
           List<Block> blocks= response.getBlocks();
           for (Block block : blocks) {
               DisplayBlockInfo(block);
           }
           paginationToken=response.getNextToken();
           if (paginationToken==null)
               finished=true;

       }

   }

   private static void StartDocumentAnalysis(String bucket, String document)
throws Exception{
       //Create notification channel
       NotificationChannel channel= new NotificationChannel()
               .withSNSTopicArn(snsTopicArn)
               .withRoleArn(roleArn);

       StartDocumentAnalysisRequest req = new StartDocumentAnalysisRequest()
               .withFeatureTypes("TABLES","FORMS")
               .withDocumentLocation(new DocumentLocation()
                   .withS3Object(new S3Object()
                       .withBucket(bucket)
                       .withName(document)))
               .withJobTag("AnalyzingText")
               .withNotificationChannel(channel);
       StartDocumentAnalysisResult startDocumentAnalysisResult =
textract.startDocumentAnalysis(req);
       startJobId=startDocumentAnalysisResult.getJobId();
   }
   //Gets the results of processing started by StartDocumentAnalysis
   private static void GetDocumentAnalysisResults() throws Exception{

       int maxResults=1000;
       String paginationToken=null;
       GetDocumentAnalysisResult response=null;
       Boolean finished=false;

       //loops until pagination token is null
       while (finished==false)
       {
           GetDocumentAnalysisRequest documentAnalysisRequest= new
GetDocumentAnalysisRequest()
                   .withJobId(startJobId)
                   .withMaxResults(maxResults)
                   .withNextToken(paginationToken);

           response = textract.getDocumentAnalysis(documentAnalysisRequest);

           DocumentMetadata documentMetaData=response.getDocumentMetadata();

           System.out.println("Pages: " + documentMetaData.getPages().toString());

           //Show blocks, confidence and detection times
           List<Block> blocks= response.getBlocks();

           for (Block block : blocks) {
               DisplayBlockInfo(block);
           }
           paginationToken=response.getNextToken();
           if (paginationToken==null)
               finished=true;
       }

   }
   //Displays Block information for text detection and text analysis
   private static void DisplayBlockInfo(Block block) {
       System.out.println("Block Id : " + block.getId());
       if (block.getText()!=null)
           System.out.println("\tDetected text: " + block.getText());
       System.out.println("\tType: " + block.getBlockType());

       if (block.getBlockType().equals("PAGE") !=true) {
           System.out.println("\tConfidence: " +
block.getConfidence().toString());
       }
       if(block.getBlockType().equals("CELL"))
       {
           System.out.println("\tCell information:");
           System.out.println("\t\tColumn: " + block.getColumnIndex());
           System.out.println("\t\tRow: " + block.getRowIndex());
           System.out.println("\t\tColumn span: " + block.getColumnSpan());
           System.out.println("\t\tRow span: " + block.getRowSpan());

       }

       System.out.println("\tRelationships");
       List<Relationship> relationships=block.getRelationships();
       if(relationships!=null) {
           for (Relationship relationship : relationships) {
               System.out.println("\t\tType: " + relationship.getType());
```

```
                System.out.println("\t\tIDs: " + relationship.getIds().toString());
            }
        } else {
            System.out.println("\t\tNo related Blocks");
        }

        System.out.println("\tGeometry");
        System.out.println("\t\tBounding Box: " +
 block.getGeometry().getBoundingBox().toString());
        System.out.println("\t\tPolygon: " +
 block.getGeometry().getPolygon().toString());

        List<String> entityTypes = block.getEntityTypes();

        System.out.println("\tEntity Types");
        if(entityTypes!=null) {
            for (String entityType : entityTypes) {
                System.out.println("\t\tEntity Type: " + entityType);
            }
        } else {
            System.out.println("\t\tNo entity type");
        }

        if(block.getBlockType().equals("SELECTION_ELEMENT")) {
            System.out.print("    Selection element detected: ");
            if (block.getSelectionStatus().equals("SELECTED")){
                System.out.println("Selected");
            }else {
                System.out.println(" Not selected");
            }
        }
        if(block.getPage()!=null)
            System.out.println("\tPage: " + block.getPage());
        System.out.println();
    }
}
```

AWS CLI

This AWS CLI command starts the asynchronous detection of text in a specified document. It returns a `job-id` that can be used to retreive the results of the detection.

```
aws textract start-document-text-detection --document-location
"{\"S3Object\":{\"Bucket\":\"bucket-name\",\"Name\":\"file-name\"}}" --
region region-name
```

This AWS CLI command returns the results for an Amazon Textract asynchronous operation when provided with a `job-id`.

```
aws textract get-document-text-detection --region region-name --job-id job-id-
number
```

If you are accessing the CLI on a Windows device, use double quotes instead of single quotes and escape the inner double quotes by backslash (i.e. \) to address any parser errors you may encounter. For an example, see below

```
aws textract start-document-text-detection --document-location "{\"S3Object\":
{\"Bucket\":\"bucket\",\"Name\":\"document\"}}" --region region-name
```

Python

```
import boto3
import json
import sys
import time


class ProcessType:
    DETECTION = 1
    ANALYSIS = 2


class DocumentProcessor:
    jobId = ''
    region_name = ''

    roleArn = ''
    bucket = ''
    document = ''

    sqsQueueUrl = ''
    snsTopicArn = ''
    processType = ''

    def __init__(self, role, bucket, document, region):
        self.roleArn = role
        self.bucket = bucket
        self.document = document
        self.region_name = region

        self.textract = boto3.client('textract', region_name=self.region_name)
        self.sqs = boto3.client('sqs')
        self.sns = boto3.client('sns')

    def ProcessDocument(self, type):
        jobFound = False

        self.processType = type
        validType = False

        # Determine which type of processing to perform
        if self.processType == ProcessType.DETECTION:
            response = self.textract.start_document_text_detection(
                DocumentLocation={'S3Object': {'Bucket': self.bucket, 'Name':
 self.document}},
                NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})
            print('Processing type: Detection')
            validType = True

        if self.processType == ProcessType.ANALYSIS:
            response = self.textract.start_document_analysis(
                DocumentLocation={'S3Object': {'Bucket': self.bucket, 'Name':
 self.document}},
                FeatureTypes=["TABLES", "FORMS"],
                NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})
            print('Processing type: Analysis')
            validType = True

        if validType == False:
            print("Invalid processing type. Choose Detection or Analysis.")
            return
        print('Start Job Id: ' + response['JobId'])
        dotLine = 0
        while jobFound == False:
            sqsResponse = self.sqs.receive_message(QueueUrl=self.sqsQueueUrl,
 MessageAttributeNames=['ALL'],
                                                   MaxNumberOfMessages=10)

            if sqsResponse:

                if 'Messages' not in sqsResponse:
                    if dotLine < 40:
                        print('.', end='')
                        dotLine = dotLine + 1
                    else:
                        print()
                        dotLine = 0
                    sys.stdout.flush()
                    time.sleep(5)
                    continue

                for message in sqsResponse['Messages']:
                    notification = json.loads(message['Body'])
                    textMessage = json.loads(notification['Message'])
                    print(textMessage['JobId'])
                    print(textMessage['Status'])
                    if str(textMessage['JobId']) == response['JobId']:
                        print('Matching Job Found:' + textMessage['JobId'])
                        jobFound = True
                        self.GetResults(textMessage['JobId'])
                        self.sqs.delete_message(QueueUrl=self.sqsQueueUrl,

 ReceiptHandle=message['ReceiptHandle'])
                    else:
                        print("Job didn't match:" +
                              str(textMessage['JobId']) + ' : ' +
 str(response['JobId']))
                    # Delete the unknown message. Consider sending to dead letter
 queue
                    self.sqs.delete_message(QueueUrl=self.sqsQueueUrl,
                                            ReceiptHandle=message['ReceiptHandle'])

        print('Done!')

    def CreateTopicandQueue(self):

        millis = str(int(round(time.time() * 1000)))

        # Create SNS topic
        snsTopicName = "AmazonTextractTopic" + millis

        topicResponse = self.sns.create_topic(Name=snsTopicName)
        self.snsTopicArn = topicResponse['TopicArn']

        # create SQS queue
        sqsQueueName = "AmazonTextractQueue" + millis
        self.sqs.create_queue(QueueName=sqsQueueName)
        self.sqsQueueUrl = self.sqs.get_queue_url(QueueName=sqsQueueName)
['QueueUrl']

        attribs = self.sqs.get_queue_attributes(QueueUrl=self.sqsQueueUrl,
                                                AttributeNames=['QueueArn'])
['Attributes']

        sqsQueueArn = attribs['QueueArn']
        # Subscribe SQS queue to SNS topic
        self.sns.subscribe(
            TopicArn=self.snsTopicArn,
            Protocol='sqs',
            Endpoint=sqsQueueArn)

        # Authorize SNS to write SQS queue
        policy = """{{
  "Version":"2012-10-17",
  "Statement":[
    {{
      "Sid":"MyPolicy",
      "Effect":"Allow",
      "Principal" : {{"AWS" : "*"}},
      "Action":"SQS:SendMessage",
      "Resource": "{}",
      "Condition":{{
        "ArnEquals":{{
          "aws:SourceArn": "{}"
        }}
      }}
    }}
  ]
}}""".format(sqsQueueArn, self.snsTopicArn)

        response = self.sqs.set_queue_attributes(
            QueueUrl=self.sqsQueueUrl,
            Attributes={
                'Policy': policy
            })

    def DeleteTopicandQueue(self):
        self.sqs.delete_queue(QueueUrl=self.sqsQueueUrl)
        self.sns.delete_topic(TopicArn=self.snsTopicArn)

    # Display information about a block
    def DisplayBlockInfo(self, block):

        print("Block Id: " + block['Id'])
        print("Type: " + block['BlockType'])
        if 'EntityTypes' in block:
            print('EntityTypes: {}'.format(block['EntityTypes']))

        if 'Text' in block:
            print("Text: " + block['Text'])

        if block['BlockType'] != 'PAGE':
            print("Confidence: " + "{:.2f}".format(block['Confidence']) + "%")

        print('Page: {}'.format(block['Page']))

        if block['BlockType'] == 'CELL':
            print('Cell Information')
            print('\tColumn: {} '.format(block['ColumnIndex']))
            print('\tRow: {}'.format(block['RowIndex']))
            print('\tColumn span: {} '.format(block['ColumnSpan']))
            print('\tRow span: {}'.format(block['RowSpan']))

            if 'Relationships' in block:
                print('\tRelationships: {}'.format(block['Relationships']))

        print('Geometry')
        print('\tBounding Box: {}'.format(block['Geometry']['BoundingBox']))
        print('\tPolygon: {}'.format(block['Geometry']['Polygon']))

        if block['BlockType'] == 'SELECTION_ELEMENT':
            print('    Selection element detected: ', end='')
            if block['SelectionStatus'] == 'SELECTED':
                print('Selected')
            else:
                print('Not selected')

    def GetResults(self, jobId):
        maxResults = 1000
        paginationToken = None
        finished = False

        while finished == False:

            response = None

            if self.processType == ProcessType.ANALYSIS:
                if paginationToken == None:
                    response = self.textract.get_document_analysis(JobId=jobId,

MaxResults=maxResults)
                else:
                    response = self.textract.get_document_analysis(JobId=jobId,

MaxResults=maxResults,

NextToken=paginationToken)

            if self.processType == ProcessType.DETECTION:
                if paginationToken == None:
                    response =
self.textract.get_document_text_detection(JobId=jobId,

MaxResults=maxResults)
                else:
                    response =
self.textract.get_document_text_detection(JobId=jobId,

MaxResults=maxResults,

NextToken=paginationToken)

            blocks = response['Blocks']
            print('Detected Document Text')
            print('Pages: {}'.format(response['DocumentMetadata']['Pages']))

            # Display block information
            for block in blocks:
                self.DisplayBlockInfo(block)
                print()
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True

    def GetResultsDocumentAnalysis(self, jobId):
        maxResults = 1000
        paginationToken = None
        finished = False

        while finished == False:

            response = None
            if paginationToken == None:
                response = self.textract.get_document_analysis(JobId=jobId,
 MaxResults=maxResults)
            else:
                response = self.textract.get_document_analysis(JobId=jobId,

 MaxResults=maxResults,

 NextToken=paginationToken)

                # Get the text blocks
            blocks = response['Blocks']
            print('Analyzed Document Text')
            print('Pages: {}'.format(response['DocumentMetadata']['Pages']))
            # Display block information
            for block in blocks:
                self.DisplayBlockInfo(block)
                print()
                print()

                if 'NextToken' in response:
                    paginationToken = response['NextToken']
                else:
                    finished = True


def main():
    roleArn = ''
    bucket = ''
    document = ''
    region_name = ''

    analyzer = DocumentProcessor(roleArn, bucket, document, region_name)
    analyzer.CreateTopicandQueue()
    analyzer.ProcessDocument(ProcessType.DETECTION)
    analyzer.DeleteTopicandQueue()


if __name__ == "__main__":
    main()
```

4. Run the code. The operation might take a while to finish. After it's finished, a list of blocks for detected or analyzed text is displayed.

Amazon Textract Results Notification

Amazon Textract publishes the results of an Amazon Textract analysis request, including completion status, to an Amazon Simple Notification Service (Amazon SNS) topic. To get the notification from an Amazon SNS topic, use an Amazon SQS queue or an AWS Lambda function. For more information, see Calling Amazon Textract Asynchronous Operations (p. 108). For an example, see Detecting or Analyzing Text in a Multipage Document (p. 119).

The results have the following JSON format:

```
{
  "JobId": "String",
  "Status": "String",
  "API": "String",
  "JobTag": "String",
  "Timestamp": Number,
  "DocumentLocation": {
    "S3ObjectName": "String",
    "S3Bucket": "String"
    }
}
```

This table describes the different parameters within an Amazon Textract response.

Parameter	Description
JobId	The unique identifier that Amazon Textract assigns to the job. It matches a job identifier that's returned from a `Start` operation, such as StartDocumentTextDetection (p. 222).
Status	The status of the job. Valid values are Succeeded, Failed, or Error.
API	The Amazon Textract operation used to analyze the input document, such as StartDocumentTextDetection (p. 222) or StartDocumentAnalysis (p. 217).
JobTag	The user-specified identifier for the job. You specify `JobTag` in a call to the `Start` operation, such as StartDocumentTextDetection (p. 222).
Timestamp	The Unix timestamp that indicates when the job finished, returned in milliseconds.
DocumentLocation	Details about the document that was processed. Includes the file name and the Amazon S3 bucket that the file is stored in.

Handling Throttled Calls and Dropped Connections

An Amazon Textract operation can fail if you exceed the maximum number of transactions per second (TPS), causing the service to throttle your application, or when your connection drops. For example, if you make too many calls to Amazon Textract operations in a short period of time, it throttles your calls and sends a `ProvisionedThroughputExceededException` error in the operation response. For information about Amazon Textract TPS quotas, see Amazon Textract Quotas.

You can manage throttling and dropped connections by automatically retrying the operation. You can specify the number of retries by including the `Config` parameter when you create the Amazon Textract client. We recommend a retry count of 5. The AWS SDK retries an operation the specified number of times before failing and throwing an exception. For more information, see Error Retries and Exponential Backoff in AWS.

> **Note**
> Automatic retries work for both synchronous and asynchronous operations. Before specifying automatic retries, make sure you have the most recent version of the AWS SDK. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 29).

The following example shows how to automatically retry Amazon Textract operations when you're processing multiple documents.

Prerequisites

- If you haven't already:

    a. Create or update an IAM user with `AmazonTextractFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 29).

    b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 29).

To automatically retry operations

1. Upload multiple document images to your S3 bucket to run the Synchronous example. Upload a multi-page document to your S3 bucket and run `StartDocumentTextDetection` on it to run the Asynchronous example.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service User Guide*.
2. The following examples demonstrate how to use the `Config` parameter to automatically retry an operation. The Synchronous example calls the `DetectDocumentText` operation, while the Asynchronous example calls the `GetDocumentTextDetection` operation.

   Sync Example

   Use the following examples to call the `DetectDocumentText` operation on the documents in your Amazon S3 bucket. In `main`, change the value of `bucket` to your S3 bucket. Change the value of `documents` to the names of the document images that you uploaded in step 2.

```
import boto3
from botocore.client import Config
# Documents
```

```
def process_multiple_documents(bucket, documents):

    config = Config(retries = dict(max_attempts = 5))

    # Amazon Textract client
    textract = boto3.client('textract', config=config)

    for documentName in documents:

        print("\nProcessing:
 {}\n==========================================".format(documentName))

        # Call Amazon Textract
        response = textract.detect_document_text(
            Document={
                'S3Object': {
                    'Bucket': bucket,
                    'Name': documentName
                }
            })

        # Print detected text
        for item in response["Blocks"]:
            if item["BlockType"] == "LINE":
                print ('\033[94m' +  item["Text"] + '\033[0m')


def main():
    bucket = ""
    documents = ["document-image-1.png",
    "document-image-2.png", "document-image-3.png",
    "document-image-4.png", "document-image-5.png" ]
    process_multiple_documents(bucket, documents)



if __name__ == "__main__":
    main()
```

Async Example

Use the following examples to call the `GetDocumentTextDetection` operation. It assumes you have already called `StartDocumentTextDetection` on the documents in your Amazon S3 bucket and obtained a `JobId`. In `main`, change the value of `bucket` to your S3 bucket and the value of `roleArn` to the Arn assigned to your Textract role. You'll also need to change the value of `document` to the name of your multi-page document in your Amazon S3 bucket. Finally, replace the value of `region_name` with the name of your region and provide the `GetResults` function with the name of your `jobId`.

```
import boto3
from botocore.client import Config

class DocumentProcessor:
    jobId = ''
    region_name = ''

    roleArn = ''
    bucket = ''
    document = ''

    sqsQueueUrl = ''
    snsTopicArn = ''
    processType = ''

    def __init__(self, role, bucket, document, region):
        self.roleArn = role
        self.bucket = bucket
        self.document = document
        self.region_name = region
        self.config = Config(retries = dict(max_attempts = 5))

        self.textract = boto3.client('textract', region_name=self.region_name,
 config=self.config)
        self.sqs = boto3.client('sqs')
        self.sns = boto3.client('sns')

# Display information about a block
    def DisplayBlockInfo(self, block):

        print("Block Id: " + block['Id'])
        print("Type: " + block['BlockType'])
        if 'EntityTypes' in block:
            print('EntityTypes: {}'.format(block['EntityTypes']))

        if 'Text' in block:
            print("Text: " + block['Text'])

        if block['BlockType'] != 'PAGE':
            print("Confidence: " + "{:.2f}".format(block['Confidence']) + "%")

        print('Page: {}'.format(block['Page']))

        if block['BlockType'] == 'CELL':
            print('Cell Information')
            print('\tColumn: {} '.format(block['ColumnIndex']))
            print('\tRow: {}'.format(block['RowIndex']))
            print('\tColumn span: {} '.format(block['ColumnSpan']))
            print('\tRow span: {}'.format(block['RowSpan']))

            if 'Relationships' in block:
                print('\tRelationships: {}'.format(block['Relationships']))

        print('Geometry')
        print('\tBounding Box: {}'.format(block['Geometry']['BoundingBox']))
        print('\tPolygon: {}'.format(block['Geometry']['Polygon']))

        if block['BlockType'] == 'SELECTION_ELEMENT':
            print('    Selection element detected: ', end='')
            if block['SelectionStatus'] == 'SELECTED':
                print('Selected')
            else:
                print('Not selected')

    def GetResults(self, jobId):
        maxResults = 1000
        paginationToken = None
        finished = False

        while finished == False:

            response = None

            if paginationToken == None:
                response = self.textract.get_document_text_detection(JobId=jobId,

 MaxResults=maxResults)
            else:
                response = self.textract.get_document_text_detection(JobId=jobId,
 MaxResults=maxResults,

 NextToken=paginationToken)

            blocks = response['Blocks']
            print('Detected Document Text')
            print('Pages: {}'.format(response['DocumentMetadata']['Pages']))

            # Display block information
            for block in blocks:
                self.DisplayBlockInfo(block)
                print()
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True

def main():
    roleArn = 'role-arn'
    bucket = 'bucket-name'
    document = 'document-name'
    region_name = 'region-name'
    analyzer = DocumentProcessor(roleArn, bucket, document, region_name)
    analyzer.GetResults("job-id")

if __name__ == "__main__":
    main()
```

Best Practices for Amazon Textract

Amazon Textract uses machine learning to read documents as a person would. It extracts text, tables, and forms from documents. Use the following best practices to get the best results from your documents.

Provide an Optimal Input Document

The following is a list of a few ways that you can optimize your input documents for better results.

- Ensure that your document text is in a language that Amazon Textract supports. Currently, Amazon Textract supports English, Spanish, German, Italian, French, and Portuguese.
- Provide a high quality image, ideally at least 150 DPI.
- If your document is already in one of the file formats that Amazon Textract supports (PDF, TIFF, JPEG, and PNG), don't convert or downsample the document before uploading it to Amazon Textract.

For the best results when extracting text from tables in documents, ensure that:

- Tables in your document are visually separated from surrounding elements on the page. For example, the table isn't overlaid onto an image or complex pattern.
- Text within the table is upright. For example, the text isn't rotated relative to other text on the page.

When extracting text from tables, you might see inconsistent results when:

- Merged table cells that span multiple columns.
- Tables with cells, rows, or columns that are different from other parts of the same table.

We recommend using text detection (p. 3) as a workaround.

Use Confidence Scores

You should take into account the confidence scores returned by Amazon Textract API operations and the sensitivity of their use case. A confidence score is a number between 0 and 100 that indicates the probability that a given prediction is correct. It helps you make informed decisions about how you use the results.

In applications that are sensitive to detection errors (false positives), enforce a minimum confidence score threshold. The application should discard results below that threshold or flag situations as requiring a higher level of human scrutiny.

The optimal threshold depends on the application. For archival purposes, such as documenting handwritten notes, it might be as low as 50%. Business processes involving financial decisions might require thresholds of 90% or higher.

Consider Using Human Review

Also consider incorporating human review into your workflows. This is especially important for sensitive applications, such as business processes that involve financial decisions.

Examples

the section called “ Block ” (p. 231) objects that are returned from Amazon Textract operations contain the results of text detection and text analysis operations, such as the section called “ AnalyzeDocument ” (p. 187). The following Python examples show some of the different ways that you can use Block objects. For example, you can export table information to a comma-separated values (CSV) file.

The examples use synchronous Amazon Textract operations that return all results. If you want to use asynchronous operations such as the section called “ StartDocumentAnalysis ” (p. 217), you need to change the example code to accommodate multiple batches of returned `Block` objects. To make use of the asynchronous operations example, ensure that you have followed the instructions given at Configuring Amazon Textract for Asynchronous Operations (p. 118).

For examples that show you other ways to use Amazon Textract, see Additional Code Examples (p. 150).

Prerequisites

Before you can run the examples in this section, you have to configure your environment.

To configure your environment

1. Create or update an IAM user with `AmazonTextractFullAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 29).
2. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 29).

Extracting Key-Value Pairs from a Form Document

The following Python example shows how to extract key-value pairs in form documents from the section called “ Block ” (p. 231) objects that are stored in a map. Block objects are returned from a call to the section called “ AnalyzeDocument ” (p. 187). For more information, see Form Data (Key-Value Pairs) (p. 13).

You use the following functions:

- `get_kv_map` – Calls AnalyzeDocument (p. 187), and stores the KEY and VALUE BLOCK objects in a map.
- `get_kv_relationship` and `find_value_block` – Constructs the key-value relationships from the map.

To extract key-value pairs from a form document

1. Configure your environment. For more information, see Prerequisites (p. 139).
2. Save the following example code to a file named *textract_python_kv_parser.py*.

```
import boto3
import sys
import re
import json
```

```
def get_kv_map(file_name):

    with open(file_name, 'rb') as file:
        img_test = file.read()
        bytes_test = bytearray(img_test)
        print('Image loaded', file_name)

    # process using image bytes
    client = boto3.client('textract')
    response = client.analyze_document(Document={'Bytes': bytes_test},
 FeatureTypes=['FORMS'])

    # Get the text blocks
    blocks=response['Blocks']


    # get key and value maps
    key_map = {}
    value_map = {}
    block_map = {}
    for block in blocks:
        block_id = block['Id']
        block_map[block_id] = block
        if block['BlockType'] == "KEY_VALUE_SET":
            if 'KEY' in block['EntityTypes']:
                key_map[block_id] = block
            else:
                value_map[block_id] = block

    return key_map, value_map, block_map


def get_kv_relationship(key_map, value_map, block_map):
    kvs = {}
    for block_id, key_block in key_map.items():
        value_block = find_value_block(key_block, value_map)
        key = get_text(key_block, block_map)
        val = get_text(value_block, block_map)
        kvs[key] = val
    return kvs


def find_value_block(key_block, value_map):
    for relationship in key_block['Relationships']:
        if relationship['Type'] == 'VALUE':
            for value_id in relationship['Ids']:
                value_block = value_map[value_id]
    return value_block


def get_text(result, blocks_map):
    text = ''
    if 'Relationships' in result:
        for relationship in result['Relationships']:
            if relationship['Type'] == 'CHILD':
                for child_id in relationship['Ids']:
                    word = blocks_map[child_id]
                    if word['BlockType'] == 'WORD':
                        text += word['Text'] + ' '
                    if word['BlockType'] == 'SELECTION_ELEMENT':
                        if word['SelectionStatus'] == 'SELECTED':
                            text += 'X '


    return text
```

```


def print_kvs(kvs):
    for key, value in kvs.items():
        print(key, ":", value)


def search_value(kvs, search_key):
    for key, value in kvs.items():
        if re.search(search_key, key, re.IGNORECASE):
            return value

def main(file_name):

    key_map, value_map, block_map = get_kv_map(file_name)

    # Get Key Value relationship
    kvs = get_kv_relationship(key_map, value_map, block_map)
    print("\n\n== FOUND KEY : VALUE pairs ===\n")
    print_kvs(kvs)

    # Start searching a key value
    while input('\n Do you want to search a value for a key? (enter "n" for exit) ') !=
 'n':
        search_key = input('\n Enter a search key:')
        print('The value is:', search_value(kvs, search_key))

if __name__ == "__main__":
    file_name = sys.argv[1]
    main(file_name)
```

3. At the command prompt, enter the following command. Replace `file` with the document image file that you want to analyze.

```
textract_python_kv_parser.py file
```

4. When you're prompted, enter a key that's in the input document. If the code detects the key, it displays the key's value.

Exporting Tables into a CSV File

These Python example show how to export tables from an image of a document into a comma-separated values (CSV) file.

The example for synchronous document analysis collects table information from a call to the section called " AnalyzeDocument " (p. 187). The example for asynchronous document analysis makes a call to the section called " StartDocumentAnalysis " (p. 217) and then retrives the results from the section called " GetDocumentAnalysis " (p. 201) as `Block` objects.

Table information is returned as the section called " Block " (p. 231) objects from a call to the section called " AnalyzeDocument " (p. 187). For more information, see Tables (p. 15). The `Block` objects are stored in a map structure that's used to export the table data into a CSV file.

Synchronous

In this example, you will use the functions:

- `get_table_csv_results` – Calls AnalyzeDocument (p. 187), and builds a map of tables that are detected in the document. Creates a CSV representation of all detected tables.
- `generate_table_csv` – Generates the CSV file for an individual table.

- `get_rows_columns_map` – Gets the rows and columns from the map.
- `get_text` – Gets the text from a cell.

To export tables into a CSV file

1. Configure your environment. For more information, see Prerequisites (p. 139).
2. Save the following example code to a file named *textract_python_table_parser.py*.

```
import webbrowser, os
import json
import boto3
import io
from io import BytesIO
import sys
from pprint import pprint


def get_rows_columns_map(table_result, blocks_map):
    rows = {}
    for relationship in table_result['Relationships']:
        if relationship['Type'] == 'CHILD':
            for child_id in relationship['Ids']:
                cell = blocks_map[child_id]
                if cell['BlockType'] == 'CELL':
                    row_index = cell['RowIndex']
                    col_index = cell['ColumnIndex']
                    if row_index not in rows:
                        # create new row
                        rows[row_index] = {}

                    # get the text value
                    rows[row_index][col_index] = get_text(cell, blocks_map)
    return rows


def get_text(result, blocks_map):
    text = ''
    if 'Relationships' in result:
        for relationship in result['Relationships']:
            if relationship['Type'] == 'CHILD':
                for child_id in relationship['Ids']:
                    word = blocks_map[child_id]
                    if word['BlockType'] == 'WORD':
                        text += word['Text'] + ' '
                    if word['BlockType'] == 'SELECTION_ELEMENT':
                        if word['SelectionStatus'] =='SELECTED':
                            text +=  'X '
    return text


def get_table_csv_results(file_name):

    with open(file_name, 'rb') as file:
        img_test = file.read()
        bytes_test = bytearray(img_test)
        print('Image loaded', file_name)

    # process using image bytes
    # get the results
    client = boto3.client('textract')

    response = client.analyze_document(Document={'Bytes': bytes_test},
 FeatureTypes=['TABLES'])
```

```
    # Get the text blocks
    blocks=response['Blocks']
    pprint(blocks)

    blocks_map = {}
    table_blocks = []
    for block in blocks:
        blocks_map[block['Id']] = block
        if block['BlockType'] == "TABLE":
            table_blocks.append(block)

    if len(table_blocks) <= 0:
        return "<b> NO Table FOUND </b>"

    csv = ''
    for index, table in enumerate(table_blocks):
        csv += generate_table_csv(table, blocks_map, index +1)
        csv += '\n\n'

    return csv

def generate_table_csv(table_result, blocks_map, table_index):
    rows = get_rows_columns_map(table_result, blocks_map)

    table_id = 'Table_' + str(table_index)

    # get cells.
    csv = 'Table: {0}\n\n'.format(table_id)

    for row_index, cols in rows.items():

        for col_index, text in cols.items():
            csv += '{}'.format(text) + ","
        csv += '\n'

    csv += '\n\n\n'
    return csv

def main(file_name):
    table_csv = get_table_csv_results(file_name)

    output_file = 'output.csv'

    # replace content
    with open(output_file, "wt") as fout:
        fout.write(table_csv)

    # show the results
    print('CSV OUTPUT FILE: ', output_file)


if __name__ == "__main__":
    file_name = sys.argv[1]
    main(file_name)
```

3. At the command prompt, enter the following command. Replace `file` with the name of the document image file that you want to analyze.

```
python textract_python_table_parser.py file
```

When you run the example, the CSV output is saved in a file named `output.csv`.

Asynchronous

In this example, you will use make use of two different scripts. The first script starts the process of asynchronoulsy analyzing documents with `StartDocumentAnalysis` and gets the `Block` information returned by `GetDocumentAnalysis`. The second script takes the returned `Block` information for each page, formats the data as a table, and saves the tables to a CSV file.

To export tables into a CSV file

1. Configure your environment. For more information, see Prerequisites (p. 139).
2. Ensure that you have followed the instructions given at see Configuring Amazon Textract for Asynchronous Operations (p. 118). The process documented on that page enables you to send and receive messages about the completion status of asynchronous jobs.
3. In the following code example, replace the value of `roleArn` with the Arn assigned to the role that you created in Step 2. Replace the value of `bucket` with the name of the S3 bucket containing your document. Replace the value of `document` with the name of the document in your S3 bucket. Replace the value of `region_name` with the name of your bucket's region.

   Save the following example code to a file named *start_doc_analysis_for_table_extraction.py.*.

```
import boto3
import json
import sys
import time

class DocumentProcessor:

    jobId = ''
    region_name = ''

    roleArn = ''
    bucket = ''
    document = ''

    sqsQueueUrl = ''
    snsTopicArn = ''
    processType = ''

    def __init__(self, role, bucket, document, region):
        self.roleArn = role
        self.bucket = bucket
        self.document = document
        self.region_name = region

        self.textract = boto3.client('textract', region_name=self.region_name)
        self.sqs = boto3.client('sqs')
        self.sns = boto3.client('sns')

    def ProcessDocument(self):

        jobFound = False

        response =
 self.textract.start_document_analysis(DocumentLocation={'S3Object': {'Bucket':
 self.bucket, 'Name': self.document}},
               FeatureTypes=["TABLES", "FORMS"], NotificationChannel={'RoleArn':
 self.roleArn, 'SNSTopicArn': self.snsTopicArn})
        print('Processing type: Analysis')

        print('Start Job Id: ' + response['JobId'])

        print('Done!')
    def CreateTopicandQueue(self):

        millis = str(int(round(time.time() * 1000)))

        # Create SNS topic
        snsTopicName = "AmazonTextractTopic" + millis

        topicResponse = self.sns.create_topic(Name=snsTopicName)
        self.snsTopicArn = topicResponse['TopicArn']

        # create SQS queue
        sqsQueueName = "AmazonTextractQueue" + millis
        self.sqs.create_queue(QueueName=sqsQueueName)
        self.sqsQueueUrl = self.sqs.get_queue_url(QueueName=sqsQueueName)
['QueueUrl']

        attribs = self.sqs.get_queue_attributes(QueueUrl=self.sqsQueueUrl,
                                                AttributeNames=['QueueArn'])
['Attributes']

        sqsQueueArn = attribs['QueueArn']

        # Subscribe SQS queue to SNS topic
        self.sns.subscribe(TopicArn=self.snsTopicArn, Protocol='sqs',
 Endpoint=sqsQueueArn)

        # Authorize SNS to write SQS queue
        policy = """{{
      "Version":"2012-10-17",
      "Statement":[
        {{
          "Sid":"MyPolicy",
          "Effect":"Allow",
          "Principal" : {{"AWS" : "*"}},
          "Action":"SQS:SendMessage",
          "Resource": "{}",
          "Condition":{{
            "ArnEquals":{{
              "aws:SourceArn": "{}"
            }}
          }}
        }}
      ]
    }}""".format(sqsQueueArn, self.snsTopicArn)

        response = self.sqs.set_queue_attributes(
            QueueUrl=self.sqsQueueUrl,
            Attributes={
                'Policy': policy
            })

def main():
    roleArn = 'YOUR ROLE ARN'
    bucket = 'YOUR S3 Bucket NAME'
    document = 'NAME OF DOCUMENT IN BUCKET'
    region_name = 'NAME OF REGION'

    analyzer = DocumentProcessor(roleArn, bucket, document, region_name)
    analyzer.CreateTopicandQueue()
    analyzer.ProcessDocument()

if __name__ == "__main__":
    main()
```

4. Run the code. The code will print a JobId. Copy this JobId down.

5. Wait for your job to finish processing, and after it has finished, copy the following code to a file named *get_doc_analysis_for_table_extraction.py*. Replace the value of `jobId` with the Job ID you copied down earlier. Replace the value of `region_name` with the name of the region associated with your Textract role.

```
import boto3
from pprint import pprint

jobId = 'YOUR JOB ID'
region_name = 'YOUR REGION NAME'

textract = boto3.client('textract', region_name=region_name)

# Display information about a block
def DisplayBlockInfo(block):
    print("Block Id: " + block['Id'])
    print("Type: " + block['BlockType'])
    if 'EntityTypes' in block:
        print('EntityTypes: {}'.format(block['EntityTypes']))

    if 'Text' in block:
        print("Text: " + block['Text'])

    if block['BlockType'] != 'PAGE':
        print("Confidence: " + "{:.2f}".format(block['Confidence']) + "%")

def GetResults(jobId):
    maxResults = 1000
    paginationToken = None
    finished = False

    while finished == False:

        response = None

        if paginationToken == None:
            response = textract.get_document_analysis(JobId=jobId,
 MaxResults=maxResults)
        else:
            response = textract.get_document_analysis(JobId=jobId,
 MaxResults=maxResults,

 NextToken=paginationToken)

        blocks = response['Blocks']
        print('Detected Document Text')
        print('Pages: {}'.format(response['DocumentMetadata']['Pages']))

        # Display block information
        for block in blocks:
            DisplayBlockInfo(block)
            print()
            print()

        if 'NextToken' in response:
            paginationToken = response['NextToken']
        else:
            finished = True

        return blocks

def get_rows_columns_map(table_result, blocks_map):
    rows = {}
    for relationship in table_result['Relationships']:
        if relationship['Type'] == 'CHILD':
            for child_id in relationship['Ids']:
                cell = blocks_map[child_id]
                if cell['BlockType'] == 'CELL':
                    row_index = cell['RowIndex']
                    col_index = cell['ColumnIndex']
                    if row_index not in rows:
                        # create new row
                        rows[row_index] = {}

                    # get the text value
                    rows[row_index][col_index] = get_text(cell, blocks_map)
    return rows


def get_text(result, blocks_map):
    text = ''
    if 'Relationships' in result:
        for relationship in result['Relationships']:
            if relationship['Type'] == 'CHILD':
                for child_id in relationship['Ids']:
                    word = blocks_map[child_id]
                    if word['BlockType'] == 'WORD':
                        text += word['Text'] + ' '
                    if word['BlockType'] == 'SELECTION_ELEMENT':
                        if word['SelectionStatus'] == 'SELECTED':
                            text += 'X '
    return text


def get_table_csv_results(blocks):

    pprint(blocks)

    blocks_map = {}
    table_blocks = []
    for block in blocks:
        blocks_map[block['Id']] = block
        if block['BlockType'] == "TABLE":
            table_blocks.append(block)

    if len(table_blocks) <= 0:
        return "<b> NO Table FOUND </b>"

    csv = ''
    for index, table in enumerate(table_blocks):
        csv += generate_table_csv(table, blocks_map, index + 1)
        csv += '\n\n'

    return csv


def generate_table_csv(table_result, blocks_map, table_index):
    rows = get_rows_columns_map(table_result, blocks_map)

    table_id = 'Table_' + str(table_index)

    # get cells.
    csv = 'Table: {0}\n\n'.format(table_id)

    for row_index, cols in rows.items():

        for col_index, text in cols.items():
            csv += '{}'.format(text) + ","
        csv += '\n'
    csv += '\n\n\n'
    return csv

response_blocks = GetResults(jobId)

table_csv = get_table_csv_results(response_blocks)
output_file = 'output_test.csv'

# replace content
with open(output_file, "wt") as fout:
    fout.write(table_csv)

# show the results
print('CSV OUTPUT FILE: ', output_file)
```

6. Run the code.

   After you have obtained you results, be sure to delete the associated SNS and SQS resources, or else you may accrue charges for them.

Creating an AWS Lambda Function

You can call Amazon Textract API operations from within an AWS Lambda function. The following instructions show how to create a Lambda function in Python that calls the section called " DetectDocumentText " (p. 197). It returns a list of the section called " Block " (p. 231) objects. To run this example, you need an Amazon S3 bucket that contains a document in PNG or JPEG format. To create the function, you use the console.

For an example that uses Lambda functions to process documents at a large scale, see Large scale document processing with Amazon Textract.

To call the DetectDocumentText operation from a Lambda function:

Step 1: Create a Lambda deployment package

1. Open a command window.
2. Enter the following commands to create a deployment package with the most recent version of the AWS SDK.

```
pip install boto3 --target python/.
zip boto3-layer.zip -r python/
```

Step 2: Create a Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Specify the following.
   - Choose **Author from scratch**.
   - For **Function name**, enter a name.

- For **Runtime**, choose **Python 3.7** or **Python 3.6**.
- For **Choose or create an execution role**, choose **Create a new role with basic Lambda permissions**.

4. Choose **Create function** to create the Lambda function.
5. Open the IAM console at https://console.aws.amazon.com/iam/.
6. In the navigation pane choose **Roles**.
7. From the resources list, choose the IAM role that Lambda created for you. The role name starts with the name of your Lambda function.
8. Choose the **Permissions** tab, then choose **Attach policies**.
9. Select the AmazonTextractFullAccess and AmazonS3ReadOnlyAccess Policies.
10. Select **Attach Policy**.

For more information, see Create a Lambda Function with the Console

Step 3: Create and add a layer

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. In the navigation pane, choose **Layers**.
3. Choose **Create layer**.
4. For **Name**, enter a name.
5. For **Description**, enter a description.
6. For **Code entry type**, choose **Upload .zip file** and select **Upload**.
7. In the dialog box, select the zip file (boto3-layer.zip), the zip you created in Step 1: Create a Lambda deployment package (p. 148).
8. For **Compatible runtimes**, choose the version of the runtime that you chose in Step 2: Create a Lambda function (p. 148).
9. Choose **Create** to create the layer.
10. Choose the navigation pane menu icon.
11. In the navigation pane, choose **Functions**.
12. In the resources list, select the function you created in Step 2: Create a Lambda function (p. 148).
13. Choose **Configuration** and in the **Designer** section, choose **Layers**(under your Lambda function name).
14. In the **Layers** section, choose **Add a layer**.
15. Choose **Select from list of runtime compatible layers**.
16. In **Compatible layers**, select the **Name** and **Version** of the layer you created in step 3.
17. Choose **Add**.

Step 4: Add python code to the Function

1. In **Designer**, choose your function.
2. In the function code editor, add the the following to the file **lambda_function.py**. Change the values of `bucket` and `document` to your bucket and document.

```
import json
import boto3

def lambda_handler(event, context):
    bucket="bucket"
    document="document"
    client = boto3.client('textract')



    #process using S3 object
    response = client.detect_document_text(
        Document={'S3Object': {'Bucket': bucket, 'Name': document}})

    #Get the text blocks
    blocks=response['Blocks']

    return {
        'statusCode': 200,
        'body': json.dumps(blocks)
    }
```

3. Choose **Save** to save your Lambda function.

Step 5: Test your Lambda

1. Select **Test**.
2. Enter a value for **Event name**.
3. Choose **Create**.
4. The output, a list of the section called “ Block ” (p. 231) objects, appears in the Execution results pane.

If the AWS Lambda function returns a timeout error, an Amazon Textract API operation call might be the cause. For information about extending the timout period for a AWS Lambda function, see AWS Lambda Function Configuration.

For information about invoking a Lambda function from your code, see Invoking AWS Lambda Functions.

Additional Code Examples

The following table provides links to more Amazon Textract code examples.

Example	Description
Amazon Textract Code Samples	Show various ways in which you can use Amazon Textract.
Large scale document processing with Amazon Textract	Shows a serverless reference architecture that processes documents at a large scale.
Amazon Textract Parser	Shows how to parse the the section called “ Block ” (p. 231) objects returned by Amazon Textract operations.
Amazon Textract Documentation Code Examples	Code examples used in this guide.
Textractor	Shows how to convert Amazon Textract output into multiple formats.

Example	Description
Generate Searchable PDF documents with Amazon Textract	Shows how to create a searchable PDF document from different types of input documents such as JPG/PNG format images and scanned PDF documents.

Using Amazon Augmented AI to Add Human Review to Amazon Textract Output

Amazon Augmented AI (Amazon A2I) is a machine learning (ML) service that makes it easy to build workflows for human review of ML analyses.

Amazon Textract is integrated with Amazon A2I. You can use it to route document analysis results that have a low score to human reviewers.

You can use Amazon Textract `AnalyzeDocument` API to extract data from forms and the Amazon A2I console to specify the conditions under which Amazon A2I routes predictions to reviewers. You set conditions based on the confidence threshold of important form keys. For example, you can send a document to a human reviewer if the key *Name* or its associated value *Jane Doe* was detected with low confidence.

Topics

Core Concepts of Amazon A2I

Review the following terms to familiarize yourself with the core concepts of Amazon A2I.

Human Review Activation Conditions

You can use Amazon A2I *activation conditions* to specify when a document is sent to humans for review and the form-content that workers are asked to review.

For example, you can set an activation condition to have Amazon Textract route forms to Amazon A2I when an important key is detected with low confidence, like *Phone Number*. In this example, human reviewers are asked to review the *Phone Number* field and value associated detected by Amazon Textract.

You can use the following activation conditions to specify when forms gets sent to humans for review:

- Trigger a human review for specific form keys based on the form key confidence score. Human reviewers are asked to review these form keys and associated values.
- Trigger a human review when specific form keys are missing. Human reviewers are asked to identify these form keys and associated values.
- Trigger human review for all form keys identified by Amazon Textract with confidence scores in a specified range.
- Randomly send a sample of forms to humans for review. Human reviewers are asked to review all form keys and values detected by Amazon Textract.

When your activation condition depends on form key confidence scores, you can use two types of prediction-confidence thresholds to trigger human review:

- **Identification confidence** – The confidence score for key-value pairs detected within a form.
- **Qualification confidence** – The confidence score for text contained within a key-value pair in a form.

If you specify a confidence threshold, Amazon A2I routes only those predictions that fall within the threshold to human reviewers. You can adjust these thresholds at any time to achieve the right balance between accuracy and cost-effectiveness. This can help you implement audits to regularly monitor prediction accuracy.

Note

You can further customize the conditions under which documents are sent to humans for review using the Amazon A2I custom task type. With this task type, you specify conditions for a human review directly in your application. For information , see Use Amazon Augmented AI with Custom Task Types in the Amazon SageMaker Developer Guide.

Human review workflow (flow definition)

You use a *human review workflow*, also referred to as a *flow definition*, to specify resources used to create your human review workflow, and to specify your activation conditions.

The resources that you specify are:

- An IAM role with permission to call Amazon A2I API operations
- An Amazon S3 bucket where you want to store the output of the human review
- Your human *work team*
- A *worker task template* which includes instructions and examples to help workers complete review task

You also use the human review workflow to specify activation conditions. For more information, see Human Review Activation Conditions (p. 152).

You can use a single human review workflow to create multiple Human loops (p. 154).

You can create a human review workflow in the SageMaker console or with the SageMaker API. For information, see Create a Human Review Workflow (p. 155).

Worker task template

You use a *worker task template* to create a worker UI that is used for your human review tasks.

The worker UI displays your documents and worker instructions. It also provides tools that workers use to complete your tasks.

You can use the SageMaker console to configure your worker task template when you create a human review workflow. For information, see Create a Human Review Workflow (p. 155).

Work team

A *work team* is a group of human workers that you send your human review tasks to.

When you create a human review workflow, you specify a single work team.

With Amazon A2I, you can use a pool of reviewers within your own organization. You can also access the workforce comprised of over 500,000 independent contractors who are already performing machine learning tasks through Amazon Mechanical Turk. Another option is to use workforce vendors that are prescreened by AWS for quality and adherence to security procedures.

Amazon A2I also provides reviewers with a web interface that consists of all the instructions and tools that they need to complete their review tasks.

For each type of workforce (private, vendor, and Mechanical Turk), you can create multiple work teams. You can use each work team in multiple human review workflows. To learn to create a workforce and work teams, see Create and Manage Workforces in the Amazon SageMaker Developer Guide.

Important
Click here to see the compliance programs that cover Amazon Augmented AI at this time. If you use Amazon Augmented AI in conjunction with other AWS services (such as Amazon Rekognition and Amazon Textract), please note that Amazon Augmented AI may not be in scope for the same compliance programs as those other services. You are responsible for how you use Amazon Augmented AI, including understanding how the service will process or store customer data, and any impact on the compliance of your data environment. You should discuss your workload objectives and goals with your AWS account team; they can help you evaluate whether the service is a good fit for your proposed use case and architecture.
Currently, Amazon Augmented AI is PCI compliant except for public and vendor workforce cases. For information regarding Amazon Augmented AI HIPAA compliance, click here.

Human loops

You use a human loop to create a human review task.

You assign a human review task to a worker on your human worker team. The worker is asked to review key-value pairs detected by Amazon Textract in your input document that you specified in your activation conditions.

For example, say a picture falls between fifty and sixty percent confidence that it contains an apple. This falls within your confidence threshold for human review and is sent to a worker. The worker checks the document for an apple, marking it as containing or not containing an apple. Then, Amazon A2I sends the document back into the workflow.

When you call Amazon Textract `AnalyzeDocument` and specify a human review workflow (flow definition) and human loop name, a human review task is created when the activation conditions specified in your human review workflow are met. These tasks are created using the resources you specify in your human review workflow.

Get Started Using Amazon A2I

The following steps help you integrate Amazon A2I into an Amazon Textract single-page document analysis task. You do the following:

1. Create a human review workflow using the Amazon A2I console (recommended for new users) or Amazon A2I API.
2. To analyze a form and include human review when necessary, use the `AnalyzeDocument` operation and specify the Amazon Resource Name (ARN) of the human review workflow. The response tells you if human review is required.
3. Monitor your human loop using the Amazon A2I console and API.
4. Review the results of human review in an Amazon S3 bucket where the results are sent.

To set up an SageMaker notebook instance and use an example notebook see *End-to-end Demo Using Amazon Textract and Augmented AI* in the *Amazon SageMaker Developer Guide*

Note
This section explains how to create a human review workflow for the Amazon A2I, Amazon Textract task type. To further customize Amazon A2I and Amazon Textract integration, you can use the Amazon A2I custom task type. With this option, you provide a custom worker task template and specify the conditions under which a document is sent for human review directly

in your application. For information, see Use Amazon Augmented AI with Custom Task Types in the *Amazon SageMaker Developer Guide*.

Topics

Create a Human Review Workflow

You can create a human review workflow using Amazon A2I console (recommended for new users) or the Amazon A2I `CreateFlowDefinition` operation.

Topics

Create a Human Review Workflow (Console)

You can either complete this example using your own document in Amazon S3, or you can download this sample document and place it in your Amazon S3 bucket.

Make sure your S3 bucket is in the same AWS Region that you are using Amazon Textract. To create a bucket, seeCreate a Bucket in the *Amazon Simple Storage Service Console User Guide*.

> **Note**
> The Amazon A2I console is embedded in the SageMaker console. To use the console, you need permissions to access the SageMaker console, and to create a work team. To get started, you can use the AmazonSageMakerFullAccess IAM managed policy which includes all of the necessary permissions to perform most actions in SageMaker. For more information, see Identity and Access Management for Amazon SageMaker in the *Amazon SageMaker Developer Guide*.

Topics

Step 1: Create a Work Team (Console)

First, create a work team in the Amazon A2I console and add yourself as a worker so that you can preview the human review task in the worker portal, work team members can look at different tasks and documents assigned to them.

To create a private workforce using worker emails (Console)

1. Open the SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, under **Ground Truth**, choose **Labeling workforces**.
3. Choose **Private**, then choose **Create private team**.
4. Choose **Invite new workers by email**.
5. For this example, enter your email address and the email address of any others who you want to be able to preview the worker portal. You can paste or type a list of up to 50 email addresses, separated by commas, into the **Email Addresses** box.

6. Enter oneorganization name and contact email.
7. Choose **Create private team**.

If you add yourself to a private work team, you receive an email from `no-reply@verificationemail.com` with login information. Use the link in this email to reset your password and log in to the worker portal. This is where your human review tasks will appear after you call `AnalyzeDocument`.

Step 2: Create a Human Review Workflow (Console)

In this step, you create an Amazon Textract human review workflow.

To create a human review workflow (console)

1. Open the Amazon A2I console at https://console.aws.amazon.com/a2i to access the **Human review workflows** page.
2. Choose **Create human review workflow**.
3. For **Name**, enter a workflow name.
4. For **S3 bucket**, choose the bucket where you want Amazon A2I to store the results of your human review tasks. If you don't choose a bucket, change that to enter the name of the bucket.
5. Under **IAM role**, select **Create a new role**. A window appears with the title **Create an IAM role**. Use this window to specify the Amazon S3 buckets to which you want this role to have access. If you do not select **Any S3 bucket**, specify the output bucket that you specified in step 4, and the bucket that contains your input document.
6. For **Task type**, choose **Textract - Key-value pair extraction.**
7. In **Amazon Textract form extraction - Conditions for invoking human review**, specify activation conditions. We recommend that you set a high confidence score threshold for at least one key in your document to trigger a human review so that you can preview a worker task in the worker portal.

   If you used the sample document provided in this walkthrough, specify activation conditions as follows:

   a. Choose **Trigger a human review for specific form keys based on the form key confidence score or when specific form keys are missing.**
   b. For **Key name**, enter **`Mail Address`**.
   c. Set the **identification confidence** threshold between *0* and *99*.
   d. Set the **qualification confidence** threshold between *0* and *99*.
   e. Choose **Trigger a human review for all form keys identified by Amazon Textract with confidence scores in a specific range.**
   f. For **`identification confidence`**, choose any number between 0 and 90.
   g. For **`qualification confidence`**, choose any number between 0 and 90.

   This triggers a human review if Amazon Textract returns a confidence score that is less than 99 for *Mail Address* and its value, or if it returns a confidence score less than 90 for any key-value pair detected in the document.
8. Under **Worker task template creation**, select **Create from a default template**.
9. For **Template name**, enter a descriptive name..
10. For **Task description**, add something similar to the following:

    ```
    Read the instructions and review the document.
    ```
11. For **Workers** choose **Private**.

12. From the menu choose the private team that you created.
13. Select **Create**.

After your human review workflow is created, it appears in the table on the **Human review workflows** page. When the **Status** is **Active**, copy and save the workflow ARN.

Create a Human Review Workflow (API)

You can create a human review workflow, or a *flow definition*, using the Amazon A2I, `CreateFlowDefinition` operation.

For this example, you can either use your own document in Amazon S3, or you can download this sample document and store it in your S3 bucket.

Make sure your Amazon S3 bucket is in the same AWS Region that you plan to use to call `AnalyzeDocument`. To create a bucket, follow the instructions in Create a Bucket in the *Amazon Simple Storage Service Console User Guide*.

Prerequisites

To use the Amazon A2I API to create a human review workflow, you must complete the following prerequisites:

- Configure an IAM role with permission to call both Amazon A2I and Amazon Textract API operations. To get started, you can attach the AWS policies, AmazonAugmentedAIFullAccess, and AmazonTextractFullAccess to an IAM role. Record the IAM role Amazon Resources Name (ARN) because you will need it later.

  For more granular permissions when using Amazon Textract, see Amazon Textract Identity-Based Policy Examples (p. 171). For Amazon A2I, see Permissions and Security in Amazon Augmented AI in the *Amazon SageMaker Developer Guide*.
- Create a private work team and record the workteam ARN. If you are a new user of Amazon A2I, follow the instructions in Step 1: Create a Work Team (Console) (p. 155).
- Create a worker task template. Follow the instructions in Create a Worker Task Template to create a template using the Amazon A2I console. When you are creating the template, choose **Textract-Form Extraction** for **Template type**. In the template, replace `s3_arn` with the Amazon S3 ARN of your document. Add additional worker instructions in `<full-instructions header="Instructions"></full-instructions>`.

  If you want to preview your template, make sure your IAM role has the permissions described in Enable Worker Task Template Previews.

  After you've created your template, record the worker task template ARN.

You use the resources you created in **Prerequisites** to configure your `CreateFlowDefinition` request. In this request, you also specify activation conditions in JSON format. To learn how to configure your activation conditions, see Use Human Loop Activation Conditions JSON Schema with Amazon Textract.

Creating a Human Review Workflow (AWS SDK for Python (Boto3))

To use this example, replace the *`red`* text with your specifications and resources.

First, encode your activation conditions into a JSON object using the following code. This triggers a human review if Amazon Textract returns a confidence score that is less than 99 for *Mail Address* and its value, or if it returns a confidence score less than 90 for any key-value pair detected in the document. If you are using the sample document provided in this example, these activation conditions create a human review task.

```
import json

humanLoopActivationConditions = json.dumps("{
                "Conditions": [
                    {
                        "ConditionType": "ImportantFormKeyConfidenceCheck",
                        "ConditionParameters": {
                            "ImportantFormKey": "Mail Address",
                            "KeyValueBlockConfidenceLessThan": 99,
                            "WordBlockConfidenceLessThan": 99
                        }
                    },
                    {
                        "ConditionType": "ImportantFormKeyConfidenceCheck",
                        "ConditionParameters": {
                            "ImportantFormKey": "*",
                            "KeyValueBlockConfidenceLessThan": 90,
                            "WordBlockConfidenceLessThan": 90
                        }
                    }
                ]
            }"
)
```

Use `humanLoopActivationConditions` to configure the `create_flow_definition` request. The following example uses the SDK for Python (Boto3) to call `create_flow_definition` in us-west-2 AWS Region. It specifies using a private work team.

```
response = client.create_flow_definition(
    FlowDefinitionName='string',
    HumanLoopRequestSource={
         'AwsManagedHumanLoopRequestSource': "AWS/Textract/AnalyzeDocument/Forms/V1"
    },
    HumanLoopActivationConfig={
        'HumanLoopActivationConditionsConfig': {
            'HumanLoopActivationConditions': humanLoopActivationConditions
        }
    },
    HumanLoopConfig={
        'WorkteamArn': "arn:aws:sagemaker:us-west-2:111122223333:workteam/private-
crowd/work-team-name",
        'HumanTaskUiArn': "arn:aws:sagemaker:us-west-2:111122223333:human-task-ui/worker-
task-template-name",
        'TaskTitle': "Add a task title",
        'TaskDescription': "Describe your task",
        'TaskCount': 1,
        'TaskAvailabilityLifetimeInSeconds': 3600,
        'TaskTimeLimitInSeconds': 86400,
        'TaskKeywords': ["Document Review", "Content Review"]
        }
    },
    OutputConfig={
        'S3OutputPath': "s3://DOC-EXAMPLE-BUCKET/prefix/",
    },
    RoleArn="arn:aws:iam::111122223333:role/role-name"
)
```

Analyze the Document

To incorporate Amazon A2I into an Amazon Textract document analysis workflow, you configure `HumanLoopConfig` in the `AnalyzeDocument` operation.

In `HumanLoopConfig`, you specify your human review workflow (flow definition) ARN in `FlowDefinitionArn`, and give your human loop a name in `HumanLoopName`.

Analyze the Document (AWS SDK for Python (Boto3))

The following example uses the SDK for Python (Boto3) to call `analyze_document` in us-west-2. Replace the *red, italicized* text with your resources. For more information, see analyze_document in the *AWS SDK for Python (Boto) API Reference*.

```
client.analyze_document(Document={'S3Object': {"Bucket": "DOC-EXAMPLE-BUCKET", "Name":
 "document-name.png"}},
         HumanLoopConfig={"FlowDefinitionArn":"arn:aws:sagemaker:us-
west-2:111122223333:flow-definition/flow-definition-name",
                          "HumanLoopName":"human-loop-name",
                          "DataAttributes":{"ContentClassifiers":
["FreeOfPersonallyIdentifiableInformation"|"FreeOfAdultContent",]}},
         FeatureTypes=["FORMS"])
```

Analyze the Document (AWS CLI)

The following example uses the AWS CLI to call `analyze_document`. These examples are compatible with AWS CLI version 2. The first is shorthand syntax, the second in JSON syntax. For more information, see analyze-document in the *AWS CLI Command Reference*.

```
aws textract analyze-document \
     --document '{"S3Object":{"Bucket":"bucket_name","Name":"file_name"}}' \
     --human-loop-config HumanLoopName="test",FlowDefinitionArn="arn:aws:sagemaker:eu-
west-1:xyz:flow-definition/
hl_name",DataAttributes='{ContentClassifiers=["FreeOfPersonallyIdentifiableInformation","FreeOfAdul
     --feature-types '["FORMS"]'
```

```
aws textract analyze-document \
     --document '{"S3Object":{"Bucket":"bucket_name","Name":"file_name"}}' \
     --human-loop-config \
          '{"HumanLoopName":"test","FlowDefinitionArn":"arn:aws:sagemaker:eu-
west-1:xyz:flow-definition/hl_name","DataAttributes": {"ContentClassifiers":
["FreeOfPersonallyIdentifiableInformation","FreeOfAdultContent"]}}' \
     --feature-types '["FORMS"]'
```

Note
Avoid whites spaces in your --human-loop-config parameter, as this can cause processing issues for your code.

The response to this request contains HumanLoopActivationOutput, which indicates whether a human loop was created, and if it was, why. If a human loop was created, this object also contains the `HumanLoopArn`.

For more information about and examples using the `AnalyzeDocument` operation, see Analyzing Document Text with Amazon Textract (p. 90).

Monitor Human Loop

You can view details about your human loop and stop an active human loop in case of an error using the Amazon A2I console and API.

View Human Loop Details

You can view your human loop status in the Amazon A2I console and by using the Amazon A2I Runtime API.

To find details about your human loop (console)

1. Open the Amazon A2I console at https://console.aws.amazon.com/a2i to access the **Human review workflows** page.
2. Choose the human review workflow that you used also to configure `HumanLoopConfig` in `AnalyzeDocument`.
3. In the **Human loops** section, choose the human loop whose details you want to view.

To find details about your human loop (API):

Use the Amazon A2I `DescribeHumanLoop` operation. Specify the human loop name you used to call `AnalyzeDocument`.

The followingSDK for Python (Boto3) example calls `describe_human_loop`.

```
response = client.describe_human_loop(HumanLoopName="human-loop-name")
```

Stop a Human Loop

After a human loop has started, you can stop it using the Amazon A2I console and API.

To stop your human loop (console)

1. Open the Amazon A2I console at https://console.aws.amazon.com/a2i to access the **Human review workflows** page.
2. Choose the human review workflow that you used to configure `HumanLoopConfig` in the `AnalyzeDocument` operation.
3. In the **Human loops** section, choose the human loop that you want to stop.
4. Choose **Stop**.

To stop your human loop (API)

Use the Amazon A2I `StopHumanLoop` operation. Specify the name of the human loop you used to call `AnalyzeDocument`.

The following SDK for Python (Boto3) example calls `stop_human_loop`.

```
response = client.stop_human_loop(HumanLoopName="human-loop-name")
```

View Output Data and Worker Metrics

When a human review task is completed by a worker, Amazon A2I stores your output data in the Amazon S3 bucket that you specified in your human review workflow.

If you use a private workforce, your output data contains worker metadata that you can use to track individual worker activity.

Find Output Data in Amazon S3

Amazon A2I uses your human review workflow name as prefix to the name of the file that stores the output data of human loops created using that human review workflow.

The path to a human loop output uses the following pattern in which *YYYY*/*MM*/*DD*/*hh*/*mm*/*ss* represents the human loop creation date with year (`YYYY`), month (`MM`), and day (`DD`) and the creation time with hour (`hh`), minute (`mm`), and second (`ss`).

```
s3://output-bucket-specified-in-flow-definition/flow-definition-
name/YYYY/MM/DD/hh/mm/ss/human-loop-name/output.json
```

To see the output for a human loop, use the Amazon A2I console.

To see human loop output

1. Open the Amazon A2I console at https://console.aws.amazon.com/a2i to access the **Human review workflows** page.
2. Choose the human review workflow that you use to configure `HumanLoopConfig` in `AnalyzeDocument`.
3. In the **Human loops** section, choose the human loop whose output you want to review.
4. Under **Output location**, choose the link to the output data.

Track Private Worker Activity

When you use a private workforce for human review tasks, the output data includes following information about the worker who completed the review:

- The `workerId`.
- In `workerMetadata`:
  - `identityProviderType` – The service used to manage the private workforce.
  - `issuer` – The Amazon Cognito user pool or OIDC Identity Provider (IdP) issuer associated with the work team assigned to this human review task.
  - `sub` – A unique identifier that refers to the worker. If you created a workforce using Amazon Cognito, you can retrieve details about this worker (such as the name or username) using this ID using Amazon Cognito. To learn how, see Managing and Searching for User Accounts in Amazon Cognito Developer Guide.

The following is an example of the output you might see if you used Amazon Cognito to create a private workforce.

```
"workerId": "a12b3cdefg4h5i67",
            "workerMetadata": {
                "identityData": {
                    "identityProviderType": "Cognito",
                    "issuer": "https://cognito-idp.aws-region.amazonaws.com/aws-
region_123456789",
                    "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee"
```

The following is an example of the output you might see if you used your own OIDC IdP to create a private workforce:

```
"workerId": "a12b3cdefg4h5i67",
            "workerMetadata": {
                "identityData": {
                    "identityProviderType": "Oidc",
                    "issuer": "https://example-oidc-ipd.com/adfs",
                    "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee"
```

To learn more about using private workforces, see Use a Private Workforce in the *Amazon SageMaker Developer Guide*.

Security in Amazon Textract

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Use the following topics to learn how to secure your Amazon Textract resources.

Topics

Data Protection in Amazon Textract

Amazon Textract conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). This ensures that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon Textract or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon Textract or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

Encryption in Amazon Textract

Data encryption refers to protecting data while in transit and at rest. You can protect your data by using Amazon S3-Managed Keys or AWS KMS key at rest, alongside standard Transport Layer Security while in transit.

Encryption at Rest

The primary method of encrypting data in Amazon Textract is server-side encryption. Input documents passed from Amazon S3 buckets are encrypted by Amazon S3 and decrypted when you access them. As long as you authenticate your request and you have access permissions, there is no difference in the way you access encrypted or unencrypted objects. For example, if you share your objects using a presigned URL, that URL works the same way for both encrypted and unencrypted objects. Additionally, when you list objects in your bucket, the `List` API returns a list of all objects, regardless of whether they are encrypted.

Amazon Textract uses two mutually exclusive methods of server-side encryption.

Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)

When you use server-side encryption with Amazon S3-Managed Keys (SSE-S3), each object is encrypted with a unique key. As an additional safeguard, this method encrypts the key itself with a master key that it regularly rotates. Amazon S3 server-side encryption uses one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256), to encrypt your data. For more information, see Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys (SSE-S3).

Server-Side Encryption with KMS keys Stored in AWS Key Management Service (SSE-KMS)

Server-side encryption with KMS keys stored in AWS Key Management Service (SSE-KMS) is similar to SSE-S3, but with some additional benefits and charges for using this service. There are separate permissions for the use of a KMS key that provides added protection against unauthorized access of your objects in Amazon S3. SSE-KMS also provides you with an audit trail that shows when your KMS key was used and by whom. Additionally, you can create and manage KMS keys or use AWS managed keys that are unique to you, your service, and your Region. For more information, see Protecting Data Using Server-Side Encryption with KMS keys Stored in AWS Key Management Service (SSE-KMS).

Encryption in Transit

For data in transit, Amazon Textract uses Transport Layer Security (TLS) to encrypt data sent between the service and the agent. Additionally, Amazon Textract uses VPC endpoints to send data between the various microservices used when Amazon Textract processes a document.

Internetwork Traffic Privacy

Amazon Textract communicates exclusively through HTTPS endpoints, which are supported in all Regions supported by Amazon Textract

Identity and Access Management for Amazon Textract

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Textract resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon Textract.

Service user – If you use the Amazon Textract service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Textract features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Textract, see Troubleshooting Amazon Textract Identity and Access (p. 173).

Service administrator – If you're in charge of Amazon Textract resources at your company, you probably have full access to Amazon Textract. It's your job to determine which Amazon Textract features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Textract, see How Amazon Textract Works with IAM (p. 169).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Textract. To view example Amazon Textract identity-based policies that you can use in IAM, see Amazon Textract Identity-Based Policy Examples (p. 171).

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Textract in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Textract Works with IAM

Before you use IAM to manage access to Amazon Textract, you should understand what IAM features are available to use with Amazon Textract. To get a high-level view of how Amazon Textract and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

Amazon Textract Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources and the conditions under which actions are allowed or denied. Amazon Textract supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Asynchronous actions in Amazon Textract require two action permissions to be given, one for Start actions and one for Get actions. Additionally, if you are using an Amazon S3 bucket to pass documents, you will need to grant your account read access.

In Amazon Textract, all policy actions start with: `textract:`. For example, to grant someone permission to run an Amazon Textract operation with the Amazon Textract `AnalyzeDocument` operation, you include the `textract:AnalyzeDocument` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon Textract defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [
      "textract:action1",
      "textract:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action.

```
"Action": "textract:Describe*"
```

For a list of Amazon Textract actions, see Actions Defined by Amazon Textract in the *IAM User Guide*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Amazon Textract does not support specifying resource ARNs in a policy.

Condition Keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Amazon Textract does not provide any service-specific condition keys, but it does support using some global condition keys. For a list of all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Examples

To view examples of Amazon Textract identity-based policies, see Amazon Textract Identity-Based Policy Examples (p. 171).

Amazon Textract Resource-Based Policies

Amazon Textract does not support resource-based policies.

Authorization Based on Amazon Textract Tags

Amazon Textract does not support tagging resources or controlling access based on tags.

Amazon Textract IAM Roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon Textract

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon Textract supports using temporary credentials.

Service-Linked Roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon Textract does not support service-linked roles.

> **Note**
> As Amazon Textract does not support service-linked roles, it does not support AWS service principals. For more information on service principals, see AWS service principals in the *IAM User Guide*

Service Roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon Textract supports service roles.

Amazon Textract Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Amazon Textract resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon Textract resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Amazon Textract quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.

- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Giving Access to Synchronous Operations in Amazon Textract

This example policy grants access to the synchronous actions in Amazon Textract to an IAM user on your AWS account.

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "textract:DetectDocumentText",
                "textract:AnalyzeDocument"
            ],
            "Resource": "*"
        }
    ]
```

Giving Access to Asynchronous Operations in Amazon Textract

The following example policy gives an IAM user on your AWS account access to all asynchronous operations used in Amazon Textract.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "textract:StartDocumentTextDetection",
                "textract:StartDocumentAnalysis",
                "textract:GetDocumentTextDetection",
                "textract:GetDocumentAnalysis"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting Amazon Textract Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Textract and IAM.

Topics

I Am Not Authorized to Perform an Action in Amazon Textract

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to run `DetectDocumentText` on a test image but does not have `textract:DetectDocumentText` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 textract:DetectDocumentText on resource: textimage.png
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `textimage.png` resource using the `textract:DetectDocumentText` action.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon Textract.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Textract. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an Administrator and Want to Allow Others to Access Amazon Textract

To allow others to access Amazon Textract, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon Textract.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I Want to Allow People Outside of My AWS Account to Access My Amazon Textract Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Textract supports these features, see How Amazon Textract Works with IAM (p. 169).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging and Monitoring

To monitor Amazon Textract, use Amazon CloudWatch. This section provides information on how to set up monitoring for Amazon Textract. It also provides reference content for Amazon Textract metrics.

Topics

Monitoring Amazon Textract

With CloudWatch, you can get metrics for individual Amazon Textract operations or global Amazon Textract metrics for your account. You can use metrics to track the health of your Amazon Textract–based solution, and set up alarms to notify you when one or more metrics fall outside a defined threshold. For example, you can see metrics for the number of server errors that have occurred. You can also see metrics for the number of times a specific Amazon Textract operation has succeeded. To see metrics, you can use Amazon CloudWatch, the AWS CLI, or the CloudWatch API.

Using CloudWatch Metrics for Amazon Textract

To use metrics, you must specify the following information:

- The metric dimension or no dimension. A *dimension* is a name-value pair that helps you to uniquely identify a metric. Amazon Textract has one dimension, named *Operation*. It provides metrics for a specific operation. If you don't specify a dimension, the metric is scoped to all Amazon Textract operations within your account.
- The metric name, such as `UserErrorCount`.

You can get monitoring data for Amazon Textract by using the AWS Management Console, the AWS CLI, or the CloudWatch API. You can also use the CloudWatch API through one of the Amazon AWS Software Development Kits (SDKs) or the CloudWatch API tools. The console displays a series of graphs based on the raw data from the CloudWatch API. Depending on your needs, you might prefer to use either the graphs displayed in the console or retrieved from the API.

The following list shows some common uses for the metrics. These are suggestions to get you started, not a comprehensive list.

How Do I?	Relevant Metrics
How do I know if my application has reached the maximum number of requests per second?	Monitor the `Sum` statistic of the `ThrottledCount` metric.
How can I monitor the request errors?	Use the `Sum` statistic of the `UserErrorCount` metric.
How can I find the total number of requests?	Use the `SampleCount` statistic of the `ResponseTime` metric. This includes any request that results in an error. If you want to see only successful operation calls, use the `SuccessfulRequestCount` metric.
How can I monitor the latency of Amazon Textract operation calls?	Use the `ResponseTime` metric.

You must have the appropriate CloudWatch permissions to monitor Amazon Textract with CloudWatch. For more information, see Authentication and Access Control for Amazon CloudWatch.

Access Amazon Textract Metrics

The following examples show how to access Amazon Textract metrics using the CloudWatch console, the AWS CLI, and the CloudWatch API.

To view metrics (console)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, choose the **All Metrics** tab, and then choose **Amazon Textract**.
3. Choose **By operation**, and then choose a metric.

   For example, choose the **StartDocumentAnalysis** metric to measure how many times asynchronous document analysis has been started.
4. Choose a value for the date range. The metric count displayed in the graph.

To view metrics for successful `StartDocumentAnalysis` operation calls that have been made over a period of time (CLI)

- Open the AWS CLI and enter the following command:

```
aws cloudwatch get-metric-statistics \
    --metric-name SuccessfulRequestCount \
    --start-time 2019-02-01T00:00:00Z \
    --period 3600 \
    --end-time 2019-03-01T00:00:00Z \
    --namespace AWS/Textract \
    --dimensions Name=Operation,Value=StartDocumentAnalysis \
    --statistics Sum
```

This example shows the successful `StartDocumentAnalysis` operation calls made over a period of time. For more information, see get-metric-statistics.

To access metrics (CloudWatch API)

- Call `GetMetricStatistics`. For more information, see the Amazon CloudWatch API Reference.

Create an Alarm

You can create a CloudWatch alarm that sends an Amazon Simple Notification Service (Amazon SNS) message when the alarm changes state. An alarm watches a single metric over a time period that you specify. It performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or an Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms don't invoke actions simply because they are in a particular state. The state must have changed and have been maintained for a specified number of time periods.

To set an alarm (console)

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and choose **Create Alarm**. This opens the **Create Alarm Wizard**.
3. Choose **Select metric**.
4. In the **All metrics** tab, choose **Textract**.
5. Choose **By Operation**, and then choose a metric.

   For example, choose **StartDocumentAnalysis** to set an alarm for a maximum number of asynchronous document analysis operations.
6. Choose the **Graphed metrics** tab.
7. For **Statistic**, choose **Sum**.
8. Choose **Select metric**.
9. Fill in the **Name** and **Description**. For **Whenever**, choose **>=**, and enter a maximum value of your choice.
10. If you want CloudWatch to send you email when the alarm state is reached, for **Whenever this alarm:**, choose **State is ALARM**. To send alarms to an existing Amazon SNS topic, for **Send notification to:**, choose an existing SNS topic. To set the name and email addresses for a new email subscription list, choose **New list**. CloudWatch saves the list and displays it in the field so you can use it to set future alarms.

    Note
    If you use **New list** to create a new Amazon SNS topic, the email addresses must be verified before the intended recipients receive notifications. Amazon SNS sends email only when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, intended recipients don't receive a notification.
11. Choose **Create Alarm**.

To set an alarm (AWS CLI)

- Open the AWS CLI and enter the following command. Change the value of the `alarm-actions` parameter to reference an Amazon SNS topic that you previously created.

```
aws cloudwatch put-metric-alarm \
    --alarm-name StartDocumentAnalysisUserErrors \
    --alarm-description "Alarm when more than 10 StartDocumentAnalysys user errors
 occur within 5 minutes" \
    --metric-name UserErrorCount \
    --namespace AWS/Textract \
    --statistic Sum \
    --period 300 \
    --threshold 10 \
    --comparison-operator GreaterThanThreshold \
    --evaluation-periods 1 \
    --unit Count \
    --dimensions Name=Operation,Value=StartDocumentAnalysis \
    --alarm-actions arn:aws:sns:us-east-1:111111111111:alarmtopic
```

This example shows how to create an alarm for when more than 10 user errors occur within 5 minutes for calls to `StartDocumentAnalysis`. For more information, see put-metric-alarm.

To set an alarm (CloudWatch API)

- Call `PutMetricAlarm`. For more information, see *Amazon CloudWatch API Reference*.

CloudWatch Metrics for Amazon Textract

This section contains information about the Amazon CloudWatch metrics and the *Operation* dimension that are available for Amazon Textract.

You can also see an aggregate view of Amazon Textract metrics from the Amazon Textract console.

CloudWatch Metrics for Amazon Textract

The following table summarizes the Amazon Textract metrics.

Metric	Description
SuccessfulRequestCount	The number of successful requests. The response code range for a successful request is 200 to 299. Unit: Count Valid statistics: `Sum,Average`
ThrottledCount	The number of throttled requests. Amazon Textract throttles a request when it receives more requests than the limit of transactions per second set for your account. If the limit set for your account is frequently exceeded, you can request a limit increase. To request an increase, see AWS Service Limits. Unit: Count Valid statistics: `Sum,Average`
ResponseTime	The time in milliseconds for Amazon Textract to compute the response.

Metric	Description
	Units: 1. Count for `Data Samples` statistics 2. Milliseconds for `Average` statistics Valid statistics: `Data Samples,Average` **Note** The `ResponseTime` metric isn't included in the Amazon Textract metric pane.
ServerErrorCount	The number of server errors. The response code range for a server error is 500 to 599. Unit: Count Valid statistics: `Sum,Average`
UserErrorCount	The number of user errors (invalid parameters, invalid image, no permission, and so on). The response code range for a user error is 400 to 499. Unit: Count Valid statistics: `Sum,Average`

CloudWatch Dimension for Amazon Textract

To retrieve operation-specific metrics, use the `AWS/Textract` namespace and provide an operation dimension. For more information about dimensions, see Dimensions in the *Amazon CloudWatch User Guide*.

Logging Amazon Textract API Calls with AWS CloudTrail

Amazon Textract is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Textract. CloudTrail captures all API calls for Amazon Textract as events. The calls captured include calls from the Amazon Textract console and code calls to the Amazon Textract API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Textract. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Textract, the IP address that the request was made from, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon Textract Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Textract, that activity is recorded in a CloudTrail event along with other AWS service events in **Event**

history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Textract, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data that's collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon Textract operations are logged by CloudTrail and are documented in the API Reference. For example, calls to the `DetectDocumentText`, `AnalyzeDocument`, and `GetDocumentText` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Request Parameters and Response Fields That Aren't Logged

For privacy purposes, certain request parameters and response fields aren't logged—for example, request image bytes or response bounding box information. Amazon S3 bucket names and file names supplied in request parameters are provided in CloudTrail log entries. No information about image bytes passed in a request is provided in a CloudTrail log. The following table shows the input parameters and response parameters that aren't logged for each Amazon Textract operation.

Operation	Request Parameters	Response Fields
AnalyzeDocument	`Bytes`	All
DetectDocumentText	`Bytes`	All
StartDocumentAnalysis	None	None
GetDocumentAnalysis	None	All
StartDocumentTextDetection	None	None
GetDocumentTextDetection	None	All

Understanding Amazon Textract Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `AnalyzeDocument` operation. The image bytes for the input `document` and the analysis results (`responseElements`) aren't logged.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111111111111:user/janedoe",
        "accountId": "111111111111",
        "accessKeyId": "AIDACKCEVSQ6C2EXAMPLE",
        "userName": "janedoe"
    },
    "eventTime": "2019-04-03T23:56:31Z",
    "eventSource": "textract.amazonaws.com",
    "eventName": "AnalyzeDocument",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "198.51.100.0",
    "userAgent": "",
    "requestParameters": {
        "document": {},
        "featureTypes": [
            "TABLES"
        ]
    },
    "responseElements": null,
    "requestID": "e387676b-d1f0-4ea7-85d6-f5a344052dce",
    "eventID": "c5db79ce-e4ea-4401-8517-784481d559f7",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111111111111"
}
```

The following example shows a CloudTrail log entry for the `StartDocumentAnalysis` operation. The log entry includes the Amazon S3 bucket name and image file name in `documentLocation`. The log also includes the operation response.

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111111111111:user/janedoe",
                "accountId": "11111111111",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "janedoe"
            },
            "eventTime": "2019-04-04T01:42:24Z",
            "eventSource": "textract.amazonaws.com",
            "eventName": "StartDocumentAnalysis",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "198.51.100.0",
            "userAgent": "",
            "requestParameters": {
                "documentLocation": {
                    "s3Object": {
                        "bucket": "bucket",
                        "name": "document.png"
                    }
                },
                "featureTypes": [
                    "TABLES"
                ]
            },
            "responseElements": {
                "jobId": "f3c718b444fa603d5d625ab967008f4b620d4650c9db8ca1cae01ef7efe51373"
            },
            "requestID": "9ae352e8-9de1-41ad-b77b-85aa348c2e82",
            "eventID": "f741bca0-c3cb-4805-82ea-baf76439deef",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111111111111"
        }

    ]
}
```

Compliance Validation for Amazon Textract

Third-party auditors assess the security and compliance of Amazon Textract as part of multiple AWS compliance programs. These include HIPAA, SOC, ISO, and PCI.

Note
If you are processing data through Textract service that is subject to PCI DSS compliance then you must opt out your account by contacting AWS Support and following the process provided to you.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon Textract is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. The security hub helps you check your compliance with security industry standards and best practices.

Resilience in Amazon Textract

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Note
Cross region transfer of data is not permitted due to the General Data Protection Regulation (GDPR).

Infrastructure Security in Amazon Textract

As a managed service, Amazon Textract is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon Textract through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Configuration and Vulnerability Analysis in Amazon Textract

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Amazon Textract and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon Textract by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Amazon Textract APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Amazon Textract APIs. Traffic between your VPC and Amazon Textract does not leave the AWS network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for Amazon Textract VPC endpoints

Before you set up an interface VPC endpoint for Amazon Textract, ensure that you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

Amazon Textract supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for Amazon Textract

You can create a VPC endpoint for the Amazon Textract service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

Create a VPC endpoint for Amazon Textract using the following service name:

- com.amazonaws.*region*.textract - For creating an endpoint for most Amazon Textract operations.
- com.amazonaws.*region*.textract-fips - For creating an endpoint for Amazon Textract that complies with the Federal Information Processing Standard (FIPS) Publication 140-2 US government standard.

If you enable private DNS for the endpoint, you can make API requests to Amazon Textract using its default DNS name for the Region, for example, `textract.us-east-1.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for Amazon Textract

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon Textract. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Amazon Textract actions

The following is an example of an endpoint policy for Amazon Textract. When attached to an endpoint, this policy grants access to the listed Amazon Textract actions for all principals on all resources.

This example policy allows access to only the operations `DetectDocumentText` and `AnalyzeDocument`. Users can still call Amazon Textract operations from outside the VPC Endpoint.

```
"Statement":[
   {
      "Principal":"*",
      "Effect":"Allow",
      "Action":[
         "textract:DetectDocumentText",
         "textract:AnalyzeDocument",
         ],
         "Resource":"*"
      }
   ]
}
```

API Reference

This section provides documentation for the Amazon Textract API operations.

Topics

Actions

The following actions are supported:

AnalyzeDocument

Analyzes an input document for relationships between detected items.

The types of information returned are as follows:

- Form data (key-value pairs). The related information is returned in two Block (p. 231) objects, each of type `KEY_VALUE_SET`: a KEY `Block` object and a VALUE `Block` object. For example, *Name: Ana Silva Carolina* contains a key and value. *Name:* is the key. *Ana Silva Carolina* is the value.
- Table and table cell data. A TABLE `Block` object contains information about a detected table. A CELL `Block` object is returned for each cell in a table.
- Lines and words of text. A LINE `Block` object contains one or more WORD `Block` objects. All lines and words that are detected in the document are returned (including text that doesn't have a relationship with the value of `FeatureTypes`).

Selection elements such as check boxes and option buttons (radio buttons) can be detected in form data and in tables. A SELECTION_ELEMENT `Block` object contains information about a selection element, including the selection status.

You can choose which type of analysis to perform by specifying the `FeatureTypes` list.

The output is returned in a list of `Block` objects.

`AnalyzeDocument` is a synchronous operation. To analyze documents asynchronously, use StartDocumentAnalysis (p. 217).

For more information, see Document Text Analysis.

Request Syntax

```
{
   "Document": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   },
   "FeatureTypes": [ "string" ],
   "HumanLoopConfig": {
      "DataAttributes": {
         "ContentClassifiers": [ "string" ]
      },
      "FlowDefinitionArn": "string",
      "HumanLoopName": "string"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

Document (p. 187)

The input document as base64-encoded bytes or an Amazon S3 object. If you use the AWS CLI to call Amazon Textract operations, you can't pass image bytes. The document must be an image in JPEG or PNG format.

If you're using an AWS SDK to call Amazon Textract, you might not need to base64-encode image bytes that are passed using the `Bytes` field.

Type: Document (p. 236) object

Required: Yes

FeatureTypes (p. 187)

A list of the types of analysis to perform. Add TABLES to the list to return information about the tables that are detected in the input document. Add FORMS to return detected form data. To perform both types of analysis, add TABLES and FORMS to `FeatureTypes`. All lines and words detected in the document are included in the response (including text that isn't related to the value of `FeatureTypes`).

Type: Array of strings

Valid Values: `TABLES | FORMS`

Required: Yes

HumanLoopConfig (p. 187)

Sets the configuration for the human in the loop workflow for analyzing documents.

Type: HumanLoopConfig (p. 245) object

Required: No

Response Syntax

```
{
   "AnalyzeDocumentModelVersion": "string",
   "Blocks": [
      {
         "BlockType": "string",
         "ColumnIndex": number,
         "ColumnSpan": number,
         "Confidence": number,
         "EntityTypes": [ "string" ],
         "Geometry": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Polygon": [
               {
                  "X": number,
                  "Y": number
               }
            ]
         },
         "Id": "string",
         "Page": number,
         "Relationships": [
            {
               "Ids": [ "string" ],
               "Type": "string"
            }
         ],
         "RowIndex": number,
         "RowSpan": number,
         "SelectionStatus": "string",
         "Text": "string",
         "TextType": "string"
      }
   ],
   "DocumentMetadata": {
      "Pages": number
   },
   "HumanLoopActivationOutput": {
      "HumanLoopActivationConditionsEvaluationResults": "string",
      "HumanLoopActivationReasons": [ "string" ],
      "HumanLoopArn": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AnalyzeDocumentModelVersion (p. 188)

The version of the model used to analyze the document.

Type: String

Blocks (p. 188)

The items that are detected and analyzed by `AnalyzeDocument`.

Type: Array of Block (p. 231) objects

DocumentMetadata (p. 188)

Metadata about the analyzed document. An example is the number of pages.

Type: DocumentMetadata (p. 238) object

HumanLoopActivationOutput (p. 188)

Shows the results of the human in the loop evaluation.

Type: HumanLoopActivationOutput (p. 244) object

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document. For more information on the document limits in Amazon Textract, see Hard Limits in Amazon Textract (p. 255).

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 10 MB. The maximum document size for asynchronous operations is 500 MB for PDF files.

HTTP Status Code: 400

HumanLoopQuotaExceededException

Indicates you have exceeded the maximum number of active human in the loop workflows available

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

UnsupportedDocumentException

The format of the input document isn't supported. Documents for synchronous operations can be in PNG or JPEG format only. Documents for asynchronous operations can be in PDF format.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AnalyzeExpense

`AnalyzeExpense` synchronously analyzes an input document for financially related relationships between text.

Information is returned as `ExpenseDocuments` and seperated as follows.

- `LineItemGroups`- A data set containing `LineItems` which store information about the lines of text, such as an item purchased and its price on a receipt.
- `SummaryFields`- Contains all other information a receipt, such as header information or the vendors name.

Request Syntax

```
{
   "Document": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

Document (p. 192)

The input document, either as bytes or as an S3 object.

You pass image bytes to an Amazon Textract API operation by using the `Bytes` property. For example, you would use the `Bytes` property to pass a document loaded from a local file system. Image bytes passed by using the `Bytes` property must be base64 encoded. Your code might not need to encode document file bytes if you're using an AWS SDK to call Amazon Textract API operations.

You pass images stored in an S3 bucket to an Amazon Textract API operation by using the `S3Object` property. Documents stored in an S3 bucket don't need to be base64 encoded.

The AWS Region for the S3 bucket that contains the S3 object must match the AWS Region that you use for Amazon Textract operations.

If you use the AWS CLI to call Amazon Textract operations, passing image bytes using the Bytes property isn't supported. You must first upload the document to an Amazon S3 bucket, and then call the operation using the S3Object property.

For Amazon Textract to process an S3 object, the user must have permission to access the S3 object.

Type: Document (p. 236) object

Required: Yes

Response Syntax

```
{
   "DocumentMetadata": {
      "Pages": number
   },
   "ExpenseDocuments": [
      {
         "ExpenseIndex": number,
         "LineItemGroups": [
            {
               "LineItemGroupIndex": number,
               "LineItems": [
                  {
                     "LineItemExpenseFields": [
                        {
                           "LabelDetection": {
                              "Confidence": number,
                              "Geometry": {
                                 "BoundingBox": {
                                    "Height": number,
                                    "Left": number,
                                    "Top": number,
                                    "Width": number
                                 },
                                 "Polygon": [
                                    {
                                       "X": number,
                                       "Y": number
                                    }
                                 ]
                              },
                              "Text": "string"
                           },
                           "PageNumber": number,
                           "Type": {
                              "Confidence": number,
                              "Text": "string"
                           },
                           "ValueDetection": {
                              "Confidence": number,
                              "Geometry": {
                                 "BoundingBox": {
                                    "Height": number,
                                    "Left": number,
                                    "Top": number,
                                    "Width": number
                                 },
                                 "Polygon": [
                                    {
                                       "X": number,
                                       "Y": number
                                    }
                                 ]
                              },
                              "Text": "string"
                           }
                        }
                     ]
                  }
               ]
            }
         ],
         "SummaryFields": [
            {
               "LabelDetection": {
                  "Confidence": number,
                  "Geometry": {
                     "BoundingBox": {
                        "Height": number,
                        "Left": number,
                        "Top": number,
                        "Width": number
                     },
                     "Polygon": [
                        {
                           "X": number,
                           "Y": number
                        }
                     ]
                  },
                  "Text": "string"
               },
               "PageNumber": number,
               "Type": {
                  "Confidence": number,
                  "Text": "string"
               },
               "ValueDetection": {
                  "Confidence": number,
                  "Geometry": {
                     "BoundingBox": {
                        "Height": number,
                        "Left": number,
                        "Top": number,
                        "Width": number
                     },
                     "Polygon": [
                        {
                           "X": number,
                           "Y": number
                        }
                     ]
                  },
                  "Text": "string"
               }
            }
         ]
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

DocumentMetadata (p. 193)

Information about the input document.

Type: DocumentMetadata (p. 238) object

ExpenseDocuments (p. 193)

The expenses detected by Amazon Textract.

Type: Array of ExpenseDocument (p. 240) objects

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document. For more information on the document limits in Amazon Textract, see Hard Limits in Amazon Textract (p. 255).

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 10 MB. The maximum document size for asynchronous operations is 500 MB for PDF files.

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

UnsupportedDocumentException

The format of the input document isn't supported. Documents for synchronous operations can be in PNG or JPEG format only. Documents for asynchronous operations can be in PDF format.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DetectDocumentText

Detects text in the input document. Amazon Textract can detect lines of text and the words that make up a line of text. The input document must be an image in JPEG or PNG format. `DetectDocumentText` returns the detected text in an array of Block (p. 231) objects.

Each document page has as an associated `Block` of type PAGE. Each PAGE `Block` object is the parent of LINE `Block` objects that represent the lines of detected text on a page. A LINE `Block` object is a parent for each word that makes up the line. Words are represented by `Block` objects of type WORD.

`DetectDocumentText` is a synchronous operation. To analyze documents asynchronously, use StartDocumentTextDetection (p. 222).

For more information, see Document Text Detection.

Request Syntax

```
{
   "Document": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

Document (p. 197)

The input document as base64-encoded bytes or an Amazon S3 object. If you use the AWS CLI to call Amazon Textract operations, you can't pass image bytes. The document must be an image in JPEG or PNG format.

If you're using an AWS SDK to call Amazon Textract, you might not need to base64-encode image bytes that are passed using the `Bytes` field.

Type: Document (p. 236) object

Required: Yes

Response Syntax

```
{
   "Blocks": [
      {
         "BlockType": "string",
         "ColumnIndex": number,
         "ColumnSpan": number,
         "Confidence": number,
         "EntityTypes": [ "string" ],
         "Geometry": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Polygon": [
               {
                  "X": number,
                  "Y": number
               }
            ]
         },
         "Id": "string",
         "Page": number,
         "Relationships": [
            {
               "Ids": [ "string" ],
               "Type": "string"
            }
         ],
         "RowIndex": number,
         "RowSpan": number,
         "SelectionStatus": "string",
         "Text": "string",
         "TextType": "string"
      }
   ],
   "DetectDocumentTextModelVersion": "string",
   "DocumentMetadata": {
      "Pages": number
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Blocks (p. 197)

An array of `Block` objects that contain the text that's detected in the document.

Type: Array of Block (p. 231) objects

DetectDocumentTextModelVersion (p. 197)

Type: String

DocumentMetadata (p. 197)

Metadata about the document. It contains the number of pages that are detected in the document.

Type: DocumentMetadata (p. 238) object

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document. For more information on the document limits in Amazon Textract, see Hard Limits in Amazon Textract (p. 255).

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 10 MB. The maximum document size for asynchronous operations is 500 MB for PDF files.

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

UnsupportedDocumentException

The format of the input document isn't supported. Documents for synchronous operations can be in PNG or JPEG format only. Documents for asynchronous operations can be in PDF format.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetDocumentAnalysis

Gets the results for an Amazon Textract asynchronous operation that analyzes text in a document.

You start asynchronous text analysis by calling StartDocumentAnalysis (p. 217), which returns a job identifier (`JobId`). When the text analysis operation finishes, Amazon Textract publishes a completion status to the Amazon Simple Notification Service (Amazon SNS) topic that's registered in the initial call to `StartDocumentAnalysis`. To get the results of the text-detection operation, first check that the status value published to the Amazon SNS topic is `SUCCEEDED`. If so, call `GetDocumentAnalysis`, and pass the job identifier (`JobId`) from the initial call to `StartDocumentAnalysis`.

`GetDocumentAnalysis` returns an array of Block (p. 231) objects. The following types of information are returned:

- Form data (key-value pairs). The related information is returned in two Block (p. 231) objects, each of type `KEY_VALUE_SET`: a KEY `Block` object and a VALUE `Block` object. For example, *Name: Ana Silva Carolina* contains a key and value. *Name:* is the key. *Ana Silva Carolina* is the value.
- Table and table cell data. A TABLE `Block` object contains information about a detected table. A CELL `Block` object is returned for each cell in a table.
- Lines and words of text. A LINE `Block` object contains one or more WORD `Block` objects. All lines and words that are detected in the document are returned (including text that doesn't have a relationship with the value of the `StartDocumentAnalysis` `FeatureTypes` input parameter).

Selection elements such as check boxes and option buttons (radio buttons) can be detected in form data and in tables. A SELECTION_ELEMENT `Block` object contains information about a selection element, including the selection status.

Use the `MaxResults` parameter to limit the number of blocks that are returned. If there are more results than specified in `MaxResults`, the value of `NextToken` in the operation response contains a pagination token for getting the next set of results. To get the next page of results, call `GetDocumentAnalysis`, and populate the `NextToken` request parameter with the token value that's returned from the previous call to `GetDocumentAnalysis`.

For more information, see Document Text Analysis.

Request Syntax

```
{
   "JobId": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 201)

A unique identifier for the text-detection job. The `JobId` is returned from `StartDocumentAnalysis`. A `JobId` value is only valid for 7 days.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: Yes

MaxResults (p. 201)

The maximum number of results to return per paginated call. The largest value that you can specify is 1,000. If you specify a value greater than 1,000, a maximum of 1,000 results is returned. The default value is 1,000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 201)

If the previous response was incomplete (because there are more blocks to retrieve), Amazon Textract returns a pagination token in the response. You can use this pagination token to retrieve the next set of blocks.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.*\S.*`

Required: No

Response Syntax

```
{
   "AnalyzeDocumentModelVersion": "string",
   "Blocks": [
      {
         "BlockType": "string",
         "ColumnIndex": number,
         "ColumnSpan": number,
         "Confidence": number,
         "EntityTypes": [ "string" ],
         "Geometry": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Polygon": [
               {
                  "X": number,
                  "Y": number
               }
            ]
         },
         "Id": "string",
         "Page": number,
         "Relationships": [
            {
               "Ids": [ "string" ],
               "Type": "string"
            }
         ],
         "RowIndex": number,
         "RowSpan": number,
         "SelectionStatus": "string",
         "Text": "string",
         "TextType": "string"
      }
   ],
   "DocumentMetadata": {
      "Pages": number
   },
   "JobStatus": "string",
   "NextToken": "string",
   "StatusMessage": "string",
   "Warnings": [
      {
         "ErrorCode": "string",
         "Pages": [ number ]
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AnalyzeDocumentModelVersion (p. 202)

Type: String

Blocks (p. 202)

The results of the text-analysis operation.

Type: Array of Block (p. 231) objects

DocumentMetadata (p. 202)

Information about a document that Amazon Textract processed. `DocumentMetadata` is returned in every page of paginated responses from an Amazon Textract video operation.

Type: DocumentMetadata (p. 238) object

JobStatus (p. 202)

The current status of the text detection job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED | PARTIAL_SUCCESS`

NextToken (p. 202)

If the response is truncated, Amazon Textract returns this token. You can use this token in the subsequent request to retrieve the next set of text detection results.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.*\S.*`

StatusMessage (p. 202)

Returns if the detection job could not be completed. Contains explanation for what error occured.

Type: String

Warnings (p. 202)

A list of warnings that occurred during the document-analysis operation.

Type: Array of Warning (p. 254) objects

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidJobIdException

An invalid job identifier was passed to GetDocumentAnalysis (p. 201) or to GetDocumentAnalysis (p. 201).

HTTP Status Code: 400

InvalidKMSKeyException

Indicates you do not have decrypt permissions with the KMS key entered, or the KMS key was entered incorrectly.

HTTP Status Code: 400

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetDocumentTextDetection

Gets the results for an Amazon Textract asynchronous operation that detects text in a document. Amazon Textract can detect lines of text and the words that make up a line of text.

You start asynchronous text detection by calling StartDocumentTextDetection (p. 222), which returns a job identifier (`JobId`). When the text detection operation finishes, Amazon Textract publishes a completion status to the Amazon Simple Notification Service (Amazon SNS) topic that's registered in the initial call to `StartDocumentTextDetection`. To get the results of the text-detection operation, first check that the status value published to the Amazon SNS topic is `SUCCEEDED`. If so, call `GetDocumentTextDetection`, and pass the job identifier (`JobId`) from the initial call to `StartDocumentTextDetection`.

`GetDocumentTextDetection` returns an array of Block (p. 231) objects.

Each document page has as an associated `Block` of type PAGE. Each PAGE `Block` object is the parent of LINE `Block` objects that represent the lines of detected text on a page. A LINE `Block` object is a parent for each word that makes up the line. Words are represented by `Block` objects of type WORD.

Use the MaxResults parameter to limit the number of blocks that are returned. If there are more results than specified in `MaxResults`, the value of `NextToken` in the operation response contains a pagination token for getting the next set of results. To get the next page of results, call `GetDocumentTextDetection`, and populate the `NextToken` request parameter with the token value that's returned from the previous call to `GetDocumentTextDetection`.

For more information, see Document Text Detection.

Request Syntax

```
{
   "JobId": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 206)

A unique identifier for the text detection job. The `JobId` is returned from `StartDocumentTextDetection`. A `JobId` value is only valid for 7 days.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: Yes

MaxResults (p. 206)

The maximum number of results to return per paginated call. The largest value you can specify is 1,000. If you specify a value greater than 1,000, a maximum of 1,000 results is returned. The default value is 1,000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 206)

If the previous response was incomplete (because there are more blocks to retrieve), Amazon Textract returns a pagination token in the response. You can use this pagination token to retrieve the next set of blocks.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.*\S.*`

Required: No

Response Syntax

```
{
   "Blocks": [
      {
         "BlockType": "string",
         "ColumnIndex": number,
         "ColumnSpan": number,
         "Confidence": number,
         "EntityTypes": [ "string" ],
         "Geometry": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Polygon": [
               {
                  "X": number,
                  "Y": number
               }
            ]
         },
         "Id": "string",
         "Page": number,
         "Relationships": [
            {
               "Ids": [ "string" ],
               "Type": "string"
            }
         ],
         "RowIndex": number,
         "RowSpan": number,
         "SelectionStatus": "string",
         "Text": "string",
         "TextType": "string"
      }
   ],
   "DetectDocumentTextModelVersion": "string",
   "DocumentMetadata": {
      "Pages": number
   },
   "JobStatus": "string",
   "NextToken": "string",
   "StatusMessage": "string",
   "Warnings": [
      {
         "ErrorCode": "string",
         "Pages": [ number ]
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Blocks (p. 207)

The results of the text-detection operation.

Type: Array of Block (p. 231) objects

DetectDocumentTextModelVersion (p. 207)

Type: String

DocumentMetadata (p. 207)

Information about a document that Amazon Textract processed. `DocumentMetadata` is returned in every page of paginated responses from an Amazon Textract video operation.

Type: DocumentMetadata (p. 238) object

JobStatus (p. 207)

The current status of the text detection job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED | PARTIAL_SUCCESS`

NextToken (p. 207)

If the response is truncated, Amazon Textract returns this token. You can use this token in the subsequent request to retrieve the next set of text-detection results.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.*\S.*`

StatusMessage (p. 207)

Returns if the detection job could not be completed. Contains explanation for what error occured.

Type: String

Warnings (p. 207)

A list of warnings that occurred during the text-detection operation for the document.

Type: Array of Warning (p. 254) objects

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidJobIdException

An invalid job identifier was passed to GetDocumentAnalysis (p. 201) or to GetDocumentAnalysis (p. 201).

HTTP Status Code: 400

InvalidKMSKeyException

Indicates you do not have decrypt permissions with the KMS key entered, or the KMS key was entered incorrectly.

HTTP Status Code: 400

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetExpenseAnalysis

Gets the results for an Amazon Textract asynchronous operation that analyzes invoices and receipts. Amazon Textract finds contact information, items purchased, and vendor name, from input invoices and receipts.

You start asynchronous invoice/receipt analysis by calling StartExpenseAnalysis (p. 226), which returns a job identifier (`JobId`). Upon completion of the invoice/receipt analysis, Amazon Textract publishes the completion status to the Amazon Simple Notification Service (Amazon SNS) topic. This topic must be registered in the initial call to `StartExpenseAnalysis`. To get the results of the invoice/receipt analysis operation, first ensure that the status value published to the Amazon SNS topic is `SUCCEEDED`. If so, call `GetExpenseAnalysis`, and pass the job identifier (`JobId`) from the initial call to `StartExpenseAnalysis`.

Use the MaxResults parameter to limit the number of blocks that are returned. If there are more results than specified in `MaxResults`, the value of `NextToken` in the operation response contains a pagination token for getting the next set of results. To get the next page of results, call `GetExpenseAnalysis`, and populate the `NextToken` request parameter with the token value that's returned from the previous call to `GetExpenseAnalysis`.

For more information, see Analyzing Invoices and Receipts.

Request Syntax

```
{
   "JobId": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 211)

A unique identifier for the text detection job. The `JobId` is returned from `StartExpenseAnalysis`. A `JobId` value is only valid for 7 days.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: Yes

MaxResults (p. 211)

The maximum number of results to return per paginated call. The largest value you can specify is 20. If you specify a value greater than 20, a maximum of 20 results is returned. The default value is 20.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 211)

If the previous response was incomplete (because there are more blocks to retrieve), Amazon Textract returns a pagination token in the response. You can use this pagination token to retrieve the next set of blocks.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.*\S.*`

Required: No

Response Syntax

```
{
   "AnalyzeExpenseModelVersion": "string",
   "DocumentMetadata": {
      "Pages": number
   },
   "ExpenseDocuments": [
      {
         "ExpenseIndex": number,
         "LineItemGroups": [
            {
               "LineItemGroupIndex": number,
               "LineItems": [
                  {
                     "LineItemExpenseFields": [
                        {
                           "LabelDetection": {
                              "Confidence": number,
                              "Geometry": {
                                 "BoundingBox": {
                                    "Height": number,
                                    "Left": number,
                                    "Top": number,
                                    "Width": number
                                 },
                                 "Polygon": [
                                    {
                                       "X": number,
                                       "Y": number
                                    }
                                 ]
                              },
                              "Text": "string"
                           },
                           "PageNumber": number,
                           "Type": {
                              "Confidence": number,
                              "Text": "string"
                           },
                           "ValueDetection": {
                              "Confidence": number,
                              "Geometry": {
                                 "BoundingBox": {
                                    "Height": number,
                                    "Left": number,
                                    "Top": number,
                                    "Width": number
                                 },
                                 "Polygon": [ 
                                    { 
                                       "X": number,
                                       "Y": number
                                    }
                                 ]
                              },
                              "Text": "string"
                           }
                        }
                     ]
                  }
               ]
            }
         ],
         "SummaryFields": [ 
            { 
               "LabelDetection": { 
                  "Confidence": number,
                  "Geometry": { 
                     "BoundingBox": { 
                        "Height": number,
                        "Left": number,
                        "Top": number,
                        "Width": number
                     },
                     "Polygon": [ 
                        { 
                           "X": number,
                           "Y": number
                        }
                     ]
                  },
                  "Text": "string"
               },
               "PageNumber": number,
               "Type": { 
                  "Confidence": number,
                  "Text": "string"
               },
               "ValueDetection": { 
                  "Confidence": number,
                  "Geometry": { 
                     "BoundingBox": { 
                        "Height": number,
                        "Left": number,
                        "Top": number,
                        "Width": number
                     },
                     "Polygon": [ 
                        { 
                           "X": number,
                           "Y": number
                        }
                     ]
                  },
                  "Text": "string"
               }
            }
         ]
      }
   ],
   "JobStatus": "string",
   "NextToken": "string",
   "StatusMessage": "string",
   "Warnings": [
      {
         "ErrorCode": "string",
         "Pages": [ number ]
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AnalyzeExpenseModelVersion (p. 212)

The current model version of AnalyzeExpense.

Type: String

DocumentMetadata (p. 212)

Information about a document that Amazon Textract processed. `DocumentMetadata` is returned in every page of paginated responses from an Amazon Textract operation.

Type: DocumentMetadata (p. 238) object

ExpenseDocuments (p. 212)

The expenses detected by Amazon Textract.

Type: Array of ExpenseDocument (p. 240) objects

JobStatus (p. 212)

The current status of the text detection job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED | PARTIAL_SUCCESS`

NextToken (p. 212)

If the response is truncated, Amazon Textract returns this token. You can use this token in the subsequent request to retrieve the next set of text-detection results.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `.*\S.*`

StatusMessage (p. 212)

Returns if the detection job could not be completed. Contains explanation for what error occured.

Type: String

Warnings (p. 212)

A list of warnings that occurred during the text-detection operation for the document.

Type: Array of Warning (p. 254) objects

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidJobIdException

An invalid job identifier was passed to GetDocumentAnalysis (p. 201) or to GetDocumentAnalysis (p. 201).

HTTP Status Code: 400

InvalidKMSKeyException

Indicates you do not have decrypt permissions with the KMS key entered, or the KMS key was entered incorrectly.

HTTP Status Code: 400

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartDocumentAnalysis

Starts the asynchronous analysis of an input document for relationships between detected items such as key-value pairs, tables, and selection elements.

`StartDocumentAnalysis` can analyze text in documents that are in JPEG, PNG, TIFF, and PDF format. The documents are stored in an Amazon S3 bucket. Use DocumentLocation (p. 237) to specify the bucket name and file name of the document.

`StartDocumentAnalysis` returns a job identifier (`JobId`) that you use to get the results of the operation. When text analysis is finished, Amazon Textract publishes a completion status to the Amazon Simple Notification Service (Amazon SNS) topic that you specify in `NotificationChannel`. To get the results of the text analysis operation, first check that the status value published to the Amazon SNS topic is `SUCCEEDED`. If so, call GetDocumentAnalysis (p. 201), and pass the job identifier (`JobId`) from the initial call to `StartDocumentAnalysis`.

For more information, see Document Text Analysis.

Request Syntax

```
{
   "ClientRequestToken": "string",
   "DocumentLocation": {
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   },
   "FeatureTypes": [ "string" ],
   "JobTag": "string",
   "KMSKeyId": "string",
   "NotificationChannel": {
      "RoleArn": "string",
      "SNSTopicArn": "string"
   },
   "OutputConfig": {
      "S3Bucket": "string",
      "S3Prefix": "string"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientRequestToken (p. 217)

The idempotent token that you use to identify the start request. If you use the same token with multiple `StartDocumentAnalysis` requests, the same `JobId` is returned. Use `ClientRequestToken` to prevent the same job from being accidentally started more than once. For more information, see Calling Amazon Textract Asynchronous Operations.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: No

DocumentLocation (p. 217)

The location of the document to be processed.

Type: DocumentLocation (p. 237) object

Required: Yes

FeatureTypes (p. 217)

A list of the types of analysis to perform. Add TABLES to the list to return information about the tables that are detected in the input document. Add FORMS to return detected form data. To perform both types of analysis, add TABLES and FORMS to `FeatureTypes`. All lines and words detected in the document are included in the response (including text that isn't related to the value of `FeatureTypes`).

Type: Array of strings

Valid Values: `TABLES | FORMS`

Required: Yes

JobTag (p. 217)

An identifier that you specify that's included in the completion notification published to the Amazon SNS topic. For example, you can use `JobTag` to identify the type of document that the completion notification corresponds to (such as a tax form or a receipt).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[a-zA-Z0-9_.\-:]+`

Required: No

KMSKeyId (p. 217)

The KMS key used to encrypt the inference results. This can be in either Key ID or Key Alias format. When a KMS key is provided, the KMS key will be used for server-side encryption of the objects in the customer bucket. When this parameter is not enabled, the result will be encrypted server side,using SSE-S3.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `^[A-Za-z0-9][A-Za-z0-9:_/+=,@.-]{0,2048}$`

Required: No

NotificationChannel (p. 217)

The Amazon SNS topic ARN that you want Amazon Textract to publish the completion status of the operation to.

Type: NotificationChannel (p. 249) object

Required: No

OutputConfig (p. 217)

Sets if the output will go to a customer defined bucket. By default, Amazon Textract will save the results internally to be accessed by the GetDocumentAnalysis operation.

Type: OutputConfig (p. 250) object

Required: No

Response Syntax

```
{
   "JobId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JobId (p. 219)

The identifier for the document text detection job. Use `JobId` to identify the job in a subsequent call to `GetDocumentAnalysis`. A `JobId` value is only valid for 7 days.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document. For more information on the document limits in Amazon Textract, see Hard Limits in Amazon Textract (p. 255).

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 10 MB. The maximum document size for asynchronous operations is 500 MB for PDF files.

HTTP Status Code: 400

IdempotentParameterMismatchException

A `ClientRequestToken` input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidKMSKeyException

Indicates you do not have decrypt permissions with the KMS key entered, or the KMS key was entered incorrectly.

HTTP Status Code: 400

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

LimitExceededException

An Amazon Textract service limit was exceeded. For example, if you start too many asynchronous jobs concurrently, calls to start operations (`StartDocumentTextDetection`, for example) raise a LimitExceededException exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Textract service limit.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

UnsupportedDocumentException

The format of the input document isn't supported. Documents for synchronous operations can be in PNG or JPEG format only. Documents for asynchronous operations can be in PDF format.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartDocumentTextDetection

Starts the asynchronous detection of text in a document. Amazon Textract can detect lines of text and the words that make up a line of text.

`StartDocumentTextDetection` can analyze text in documents that are in JPEG, PNG, TIFF, and PDF format. The documents are stored in an Amazon S3 bucket. Use DocumentLocation (p. 237) to specify the bucket name and file name of the document.

`StartTextDetection` returns a job identifier (`JobId`) that you use to get the results of the operation. When text detection is finished, Amazon Textract publishes a completion status to the Amazon Simple Notification Service (Amazon SNS) topic that you specify in `NotificationChannel`. To get the results of the text detection operation, first check that the status value published to the Amazon SNS topic is `SUCCEEDED`. If so, call GetDocumentTextDetection (p. 206), and pass the job identifier (`JobId`) from the initial call to `StartDocumentTextDetection`.

For more information, see Document Text Detection.

Request Syntax

```
{
   "ClientRequestToken": "string",
   "DocumentLocation": {
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   },
   "JobTag": "string",
   "KMSKeyId": "string",
   "NotificationChannel": {
      "RoleArn": "string",
      "SNSTopicArn": "string"
   },
   "OutputConfig": {
      "S3Bucket": "string",
      "S3Prefix": "string"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientRequestToken (p. 222)

The idempotent token that's used to identify the start request. If you use the same token with multiple `StartDocumentTextDetection` requests, the same `JobId` is returned. Use `ClientRequestToken` to prevent the same job from being accidentally started more than once. For more information, see Calling Amazon Textract Asynchronous Operations.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: No

DocumentLocation (p. 222)

The location of the document to be processed.

Type: DocumentLocation (p. 237) object

Required: Yes

JobTag (p. 222)

An identifier that you specify that's included in the completion notification published to the Amazon SNS topic. For example, you can use `JobTag` to identify the type of document that the completion notification corresponds to (such as a tax form or a receipt).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[a-zA-Z0-9_.\-:]+`

Required: No

KMSKeyId (p. 222)

The KMS key used to encrypt the inference results. This can be in either Key ID or Key Alias format. When a KMS key is provided, the KMS key will be used for server-side encryption of the objects in the customer bucket. When this parameter is not enabled, the result will be encrypted server side,using SSE-S3.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `^[A-Za-z0-9][A-Za-z0-9:_/+=,@.-]{0,2048}$`

Required: No

NotificationChannel (p. 222)

The Amazon SNS topic ARN that you want Amazon Textract to publish the completion status of the operation to.

Type: NotificationChannel (p. 249) object

Required: No

OutputConfig (p. 222)

Sets if the output will go to a customer defined bucket. By default Amazon Textract will save the results internally to be accessed with the GetDocumentTextDetection operation.

Type: OutputConfig (p. 250) object

Required: No

Response Syntax

```
{
   "JobId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JobId (p. 223)

The identifier of the text detection job for the document. Use `JobId` to identify the job in a subsequent call to `GetDocumentTextDetection`. A `JobId` value is only valid for 7 days.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document. For more information on the document limits in Amazon Textract, see Hard Limits in Amazon Textract (p. 255).

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 10 MB. The maximum document size for asynchronous operations is 500 MB for PDF files.

HTTP Status Code: 400

IdempotentParameterMismatchException

A `ClientRequestToken` input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidKMSKeyException

Indicates you do not have decrypt permissions with the KMS key entered, or the KMS key was entered incorrectly.

HTTP Status Code: 400

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

LimitExceededException

An Amazon Textract service limit was exceeded. For example, if you start too many asynchronous jobs concurrently, calls to start operations (`StartDocumentTextDetection`, for example) raise a LimitExceededException exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Textract service limit.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

UnsupportedDocumentException

The format of the input document isn't supported. Documents for synchronous operations can be in PNG or JPEG format only. Documents for asynchronous operations can be in PDF format.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartExpenseAnalysis

Starts the asynchronous analysis of invoices or receipts for data like contact information, items purchased, and vendor names.

`StartExpenseAnalysis` can analyze text in documents that are in JPEG, PNG, and PDF format. The documents must be stored in an Amazon S3 bucket. Use the DocumentLocation (p. 237) parameter to specify the name of your S3 bucket and the name of the document in that bucket.

`StartExpenseAnalysis` returns a job identifier (`JobId`) that you will provide to `GetExpenseAnalysis` to retrieve the results of the operation. When the analysis of the input invoices/ receipts is finished, Amazon Textract publishes a completion status to the Amazon Simple Notification Service (Amazon SNS) topic that you provide to the `NotificationChannel`. To obtain the results of the invoice and receipt analysis operation, ensure that the status value published to the Amazon SNS topic is `SUCCEEDED`. If so, call GetExpenseAnalysis (p. 211), and pass the job identifier (`JobId`) that was returned by your call to `StartExpenseAnalysis`.

For more information, see Analyzing Invoices and Receipts.

Request Syntax

```
{
   "ClientRequestToken": "string",
   "DocumentLocation": {
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   },
   "JobTag": "string",
   "KMSKeyId": "string",
   "NotificationChannel": {
      "RoleArn": "string",
      "SNSTopicArn": "string"
   },
   "OutputConfig": {
      "S3Bucket": "string",
      "S3Prefix": "string"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientRequestToken (p. 226)

The idempotent token that's used to identify the start request. If you use the same token with multiple `StartDocumentTextDetection` requests, the same `JobId` is returned. Use `ClientRequestToken` to prevent the same job from being accidentally started more than once. For more information, see Calling Amazon Textract Asynchronous Operations

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: No

DocumentLocation (p. 226)

The location of the document to be processed.

Type: DocumentLocation (p. 237) object

Required: Yes

JobTag (p. 226)

An identifier you specify that's included in the completion notification published to the Amazon SNS topic. For example, you can use `JobTag` to identify the type of document that the completion notification corresponds to (such as a tax form or a receipt).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[a-zA-Z0-9_.\-:]+`

Required: No

KMSKeyId (p. 226)

The KMS key used to encrypt the inference results. This can be in either Key ID or Key Alias format. When a KMS key is provided, the KMS key will be used for server-side encryption of the objects in the customer bucket. When this parameter is not enabled, the result will be encrypted server side,using SSE-S3.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `^[A-Za-z0-9][A-Za-z0-9:_/+=,@.-]{0,2048}$`

Required: No

NotificationChannel (p. 226)

The Amazon SNS topic ARN that you want Amazon Textract to publish the completion status of the operation to.

Type: NotificationChannel (p. 249) object

Required: No

OutputConfig (p. 226)

Sets if the output will go to a customer defined bucket. By default, Amazon Textract will save the results internally to be accessed by the `GetExpenseAnalysis` operation.

Type: OutputConfig (p. 250) object

Required: No

Response Syntax

```
{
   "JobId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JobId (p. 227)

A unique identifier for the text detection job. The `JobId` is returned from `StartExpenseAnalysis`. A `JobId` value is only valid for 7 days.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Errors

AccessDeniedException

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document. For more information on the document limits in Amazon Textract, see Hard Limits in Amazon Textract (p. 255).

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 10 MB. The maximum document size for asynchronous operations is 500 MB for PDF files.

HTTP Status Code: 400

IdempotentParameterMismatchException

A `ClientRequestToken` input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

HTTP Status Code: 400

InternalServerError

Amazon Textract experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidKMSKeyException

Indicates you do not have decrypt permissions with the KMS key entered, or the KMS key was entered incorrectly.

HTTP Status Code: 400

InvalidParameterException

An input parameter violated a constraint. For example, in synchronous operations, an `InvalidParameterException` exception occurs when neither of the `S3Object` or `Bytes` values are supplied in the `Document` request parameter. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Textract is unable to access the S3 object that's specified in the request. for more information, Configure Access to Amazon S3 For troubleshooting information, see Troubleshooting Amazon S3

HTTP Status Code: 400

LimitExceededException

An Amazon Textract service limit was exceeded. For example, if you start too many asynchronous jobs concurrently, calls to start operations (`StartDocumentTextDetection`, for example) raise a LimitExceededException exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Textract service limit.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Textract.

HTTP Status Code: 400

ThrottlingException

Amazon Textract is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

UnsupportedDocumentException

The format of the input document isn't supported. Documents for synchronous operations can be in PNG or JPEG format only. Documents for asynchronous operations can be in PDF format.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Data Types

The following data types are supported:

Block

A `Block` represents items that are recognized in a document within a group of pixels close to each other. The information returned in a `Block` object depends on the type of operation. In text detection for documents (for example DetectDocumentText (p. 197)), you get information about the detected words and lines of text. In text analysis (for example AnalyzeDocument (p. 187)), you can also get information about the fields, tables, and selection elements that are detected in the document.

An array of `Block` objects is returned by both synchronous and asynchronous operations. In synchronous operations, such as DetectDocumentText (p. 197), the array of `Block` objects is the entire set of results. In asynchronous operations, such as GetDocumentAnalysis (p. 201), the array is returned over one or more responses.

For more information, see How Amazon Textract Works.

Contents

BlockType

The type of text item that's recognized. In operations for text detection, the following types are returned:

- *PAGE* - Contains a list of the LINE `Block` objects that are detected on a document page.
- *WORD* - A word detected on a document page. A word is one or more ISO basic Latin script characters that aren't separated by spaces.
- *LINE* - A string of tab-delimited, contiguous words that are detected on a document page.

In text analysis operations, the following types are returned:

- *PAGE* - Contains a list of child `Block` objects that are detected on a document page.
- *KEY_VALUE_SET* - Stores the KEY and VALUE `Block` objects for linked text that's detected on a document page. Use the `EntityType` field to determine if a KEY_VALUE_SET object is a KEY `Block` object or a VALUE `Block` object.
- *WORD* - A word that's detected on a document page. A word is one or more ISO basic Latin script characters that aren't separated by spaces.
- *LINE* - A string of tab-delimited, contiguous words that are detected on a document page.
- *TABLE* - A table that's detected on a document page. A table is grid-based information with two or more rows or columns, with a cell span of one row and one column each.
- *CELL* - A cell within a detected table. The cell is the parent of the block that contains the text in the cell.
- *SELECTION_ELEMENT* - A selection element such as an option button (radio button) or a check box that's detected on a document page. Use the value of `SelectionStatus` to determine the status of the selection element.

Type: String

Valid Values: `KEY_VALUE_SET | PAGE | LINE | WORD | TABLE | CELL | SELECTION_ELEMENT`

Required: No

ColumnIndex

The column in which a table cell appears. The first column position is 1. `ColumnIndex` isn't returned by `DetectDocumentText` and `GetDocumentTextDetection`.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

ColumnSpan

The number of columns that a table cell spans. Currently this value is always 1, even if the number of columns spanned is greater than 1. `ColumnSpan` isn't returned by `DetectDocumentText` and `GetDocumentTextDetection`.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Confidence

The confidence score that Amazon Textract has in the accuracy of the recognized text and the accuracy of the geometry points around the recognized text.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

EntityTypes

The type of entity. The following can be returned:

- *KEY* - An identifier for a field on the document.
- *VALUE* - The field text.

`EntityTypes` isn't returned by `DetectDocumentText` and `GetDocumentTextDetection`.

Type: Array of strings

Valid Values: `KEY | VALUE`

Required: No

Geometry

The location of the recognized text on the image. It includes an axis-aligned, coarse bounding box that surrounds the text, and a finer-grain polygon for more accurate spatial information.

Type: Geometry (p. 243) object

Required: No

Id

The identifier for the recognized text. The identifier is only unique for a single operation.

Type: String

Pattern: `.*\S.*`

Required: No

Page

The page on which a block was detected. `Page` is returned by asynchronous operations. Page values greater than 1 are only returned for multipage documents that are in PDF or TIFF format. A scanned image (JPEG/PNG), even if it contains multiple document pages, is considered to be a single-page document. The value of `Page` is always 1. Synchronous operations don't return `Page` because every input document is considered to be a single-page document.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Relationships

A list of child blocks of the current block. For example, a LINE object has child blocks for each WORD block that's part of the line of text. There aren't Relationship objects in the list for relationships that don't exist, such as when the current block has no child blocks. The list size can be the following:

- 0 - The block has no child blocks.
- 1 - The block has child blocks.

Type: Array of Relationship (p. 252) objects

Required: No

RowIndex

The row in which a table cell is located. The first row position is 1. `RowIndex` isn't returned by `DetectDocumentText` and `GetDocumentTextDetection`.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

RowSpan

The number of rows that a table cell spans. Currently this value is always 1, even if the number of rows spanned is greater than 1. `RowSpan` isn't returned by `DetectDocumentText` and `GetDocumentTextDetection`.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

SelectionStatus

The selection status of a selection element, such as an option button or check box.

Type: String

Valid Values: `SELECTED | NOT_SELECTED`

Required: No

Text

The word or line of text that's recognized by Amazon Textract.

Type: String

Required: No

TextType

The kind of text that Amazon Textract has detected. Can check for handwritten text and printed text.

Type: String

Valid Values: `HANDWRITING | PRINTED`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

BoundingBox

The bounding box around the detected page, text, key-value pair, table, table cell, or selection element on a document page. The `left` (x-coordinate) and `top` (y-coordinate) are coordinates that represent the top and left sides of the bounding box. Note that the upper-left corner of the image is the origin (0,0).

The `top` and `left` values returned are ratios of the overall document page size. For example, if the input image is 700 x 200 pixels, and the top-left coordinate of the bounding box is 350 x 50 pixels, the API returns a `left` value of 0.5 (350/700) and a `top` value of 0.25 (50/200).

The `width` and `height` values represent the dimensions of the bounding box as a ratio of the overall document page dimension. For example, if the document page size is 700 x 200 pixels, and the bounding box width is 70 pixels, the width returned is 0.1.

Contents

Height

The height of the bounding box as a ratio of the overall document page height.

Type: Float

Required: No

Left

The left coordinate of the bounding box as a ratio of overall document page width.

Type: Float

Required: No

Top

The top coordinate of the bounding box as a ratio of overall document page height.

Type: Float

Required: No

Width

The width of the bounding box as a ratio of the overall document page width.

Type: Float

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Document

The input document, either as bytes or as an S3 object.

You pass image bytes to an Amazon Textract API operation by using the `Bytes` property. For example, you would use the `Bytes` property to pass a document loaded from a local file system. Image bytes passed by using the `Bytes` property must be base64 encoded. Your code might not need to encode document file bytes if you're using an AWS SDK to call Amazon Textract API operations.

You pass images stored in an S3 bucket to an Amazon Textract API operation by using the `S3Object` property. Documents stored in an S3 bucket don't need to be base64 encoded.

The AWS Region for the S3 bucket that contains the S3 object must match the AWS Region that you use for Amazon Textract operations.

If you use the AWS CLI to call Amazon Textract operations, passing image bytes using the Bytes property isn't supported. You must first upload the document to an Amazon S3 bucket, and then call the operation using the S3Object property.

For Amazon Textract to process an S3 object, the user must have permission to access the S3 object.

Contents

Bytes

A blob of base64-encoded document bytes. The maximum size of a document that's provided in a blob of bytes is 10 MB. The document bytes must be in PNG or JPEG format.

If you're using an AWS SDK to call Amazon Textract, you might not need to base64-encode image bytes passed using the `Bytes` field.

Type: Base64-encoded binary data object

Length Constraints: Minimum length of 1. Maximum length of 10485760.

Required: No

S3Object

Identifies an S3 object as the document source. The maximum size of a document that's stored in an S3 bucket is 10 MB for synchronous API actions and 500MB for asynchronous API actions.

Type: S3Object (p. 253) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DocumentLocation

The Amazon S3 bucket that contains the document to be processed. It's used by asynchronous operations such as StartDocumentTextDetection (p. 222).

The input document can be an image file in JPEG or PNG format. It can also be a file in PDF format.

Contents

S3Object

The Amazon S3 bucket that contains the input document.

Type: S3Object (p. 253) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DocumentMetadata

Information about the input document.

Contents

Pages

The number of pages that are detected in the document.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ExpenseDetection

An object used to store information about the Value or Label detected by Amazon Textract.

Contents

Confidence

The confidence in detection, as a percentage

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Geometry

Information about where the following items are located on a document page: detected page, text, key-value pairs, tables, table cells, and selection elements. For more information, see Item Location on a Document Page.

Type: Geometry (p. 243) object

Required: No

Text

The word or line of text recognized by Amazon Textract

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ExpenseDocument

The structure holding all the information returned by AnalyzeExpense

Contents

ExpenseIndex

Denotes which invoice or receipt in the document the information is coming from. First document will be 1, the second 2, and so on.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

LineItemGroups

Information detected on each table of a document, seperated into `LineItems`.

Type: Array of LineItemGroup (p. 248) objects

Required: No

SummaryFields

Any information found outside of a table by Amazon Textract.

Type: Array of ExpenseField (p. 241) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ExpenseField

Breakdown of detected information, seperated into the catagories Type, LabelDetection, and ValueDetection

Contents

LabelDetection

The explicitly stated label of a detected element.

Type: ExpenseDetection (p. 239) object

Required: No

PageNumber

The page number the value was detected on.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Type

The implied label of a detected element. Present alongside LabelDetection for explicit elements.

Type: ExpenseType (p. 242) object

Required: No

ValueDetection

The value of a detected element. Present in explicit and implicit elements.

Type: ExpenseDetection (p. 239) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ExpenseType

An object used to store information about the Type detected by Amazon Textract. Supported types include `LINE`, `ITEM`, `QUANTITY`, and `EXPENSE_ROW`

Contents

Confidence

The confidence of accuracy, as a percentage.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Text

The word or line of text detected by Amazon Textract.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Geometry

Information about where the following items are located on a document page: detected page, text, key-value pairs, tables, table cells, and selection elements. For more information, see Item Location on a Document Page.

Contents

BoundingBox

An axis-aligned coarse representation of the location of the recognized item on the document page.

Type: BoundingBox (p. 235) object

Required: No

Polygon

Within the bounding box, a fine-grained polygon around the recognized item.

Type: Array of Point (p. 251) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

HumanLoopActivationOutput

Shows the results of the human in the loop evaluation. If there is no HumanLoopArn, the input did not trigger human review.

Contents

HumanLoopActivationConditionsEvaluationResults

Shows the result of condition evaluations, including those conditions which activated a human review.

Type: String

Length Constraints: Maximum length of 10240.

Required: No

HumanLoopActivationReasons

Shows if and why human review was needed.

Type: Array of strings

Array Members: Minimum number of 1 item.

Required: No

HumanLoopArn

The Amazon Resource Name (ARN) of the HumanLoop created.

Type: String

Length Constraints: Maximum length of 256.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

HumanLoopConfig

Sets up the human review workflow the document will be sent to if one of the conditions is met. You can also set certain attributes of the image before review.

Contents

DataAttributes

Sets attributes of the input data.

Type: HumanLoopDataAttributes (p. 246) object

Required: No

FlowDefinitionArn

The Amazon Resource Name (ARN) of the flow definition.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

HumanLoopName

The name of the human workflow used for this image. This should be kept unique within a region.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 63.

Pattern: `^[a-z0-9](-*[a-z0-9])*`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

HumanLoopDataAttributes

Allows you to set attributes of the image. Currently, you can declare an image as free of personally identifiable information and adult content.

Contents

ContentClassifiers

Sets whether the input image is free of personally identifiable information or adult content.

Type: Array of strings

Array Members: Maximum number of 256 items.

Valid Values: `FreeOfPersonallyIdentifiableInformation | FreeOfAdultContent`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LineItemFields

A structure that holds information about the different lines found in a document's tables.

Contents

LineItemExpenseFields

ExpenseFields used to show information from detected lines on a table.

Type: Array of ExpenseField (p. 241) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LineItemGroup

A grouping of tables which contain LineItems, with each table identified by the table's `LineItemGroupIndex`.

Contents

LineItemGroupIndex

The number used to identify a specific table in a document. The first table encountered will have a LineItemGroupIndex of 1, the second 2, etc.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

LineItems

The breakdown of information on a particular line of a table.

Type: Array of LineItemFields (p. 247) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

NotificationChannel

The Amazon Simple Notification Service (Amazon SNS) topic to which Amazon Textract publishes the completion status of an asynchronous document operation, such as StartDocumentTextDetection (p. 222).

Contents

RoleArn

The Amazon Resource Name (ARN) of an IAM role that gives Amazon Textract publishing permissions to the Amazon SNS topic.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:([a-z\d-]+):iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+`

Required: Yes

SNSTopicArn

The Amazon SNS topic that Amazon Textract posts the completion status to.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1024.

Pattern: `(^arn:([a-z\d-]+):sns:[a-zA-Z\d-]{1,20}:\w{12}:.+$)`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

OutputConfig

Sets whether or not your output will go to a user created bucket. Used to set the name of the bucket, and the prefix on the output file.

`OutputConfig` is an optional parameter which lets you adjust where your output will be placed. By default, Amazon Textract will store the results internally and can only be accessed by the Get API operations. With OutputConfig enabled, you can set the name of the bucket the output will be sent to and the file prefix of the results where you can download your results. Additionally, you can set the `KMSKeyID` parameter to a customer master key (CMK) to encrypt your output. Without this parameter set Amazon Textract will encrypt server-side using the AWS managed CMK for Amazon S3.

Decryption of Customer Content is necessary for processing of the documents by Amazon Textract. If your account is opted out under an AI services opt out policy then all unencrypted Customer Content is immediately and permanently deleted after the Customer Content has been processed by the service. No copy of of the output is retained by Amazon Textract. For information about how to opt out, see Managing AI services opt-out policy.

For more information on data privacy, see the Data Privacy FAQ.

Contents

S3Bucket

The name of the bucket your output will go to.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[0-9A-Za-z\.\-_]*`

Required: Yes

S3Prefix

The prefix of the object key that the output will be saved to. When not enabled, the prefix will be “textract_output".

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `.*\S.*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Point

The X and Y coordinates of a point on a document page. The X and Y values that are returned are ratios of the overall document page size. For example, if the input document is 700 x 200 and the operation returns X=0.5 and Y=0.25, then the point is at the (350,50) pixel coordinate on the document page.

An array of `Point` objects, `Polygon`, is returned as part of the Geometry (p. 243) object that's returned in a Block (p. 231) object. A `Polygon` object represents a fine-grained polygon around detected text, a key-value pair, a table, a table cell, or a selection element.

Contents

X

The value of the X coordinate for a point on a `Polygon`.

Type: Float

Required: No

Y

The value of the Y coordinate for a point on a `Polygon`.

Type: Float

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Relationship

Information about how blocks are related to each other. A `Block` object contains 0 or more `Relation` objects in a list, `Relationships`. For more information, see Block (p. 231).

The `Type` element provides the type of the relationship for all blocks in the `IDs` array.

Contents

Ids

An array of IDs for related blocks. You can get the type of the relationship from the `Type` element.

Type: Array of strings

Pattern: `.*\S.*`

Required: No

Type

The type of relationship that the blocks in the IDs array have with the current block. The relationship can be `VALUE` or `CHILD`. A relationship of type VALUE is a list that contains the ID of the VALUE block that's associated with the KEY of a key-value pair. A relationship of type CHILD is a list of IDs that identify WORD blocks in the case of lines Cell blocks in the case of Tables, and WORD blocks in the case of Selection Elements.

Type: String

Valid Values: `VALUE | CHILD | COMPLEX_FEATURES`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

S3Object

The S3 bucket name and file name that identifies the document.

The AWS Region for the S3 bucket that contains the document must match the Region that you use for Amazon Textract operations.

For Amazon Textract to process a file in an S3 bucket, the user must have permission to access the S3 bucket and file.

Contents

Bucket

The name of the S3 bucket. Note that the # character is not valid in the file name.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[0-9A-Za-z\.\-_]*`

Required: No

Name

The file name of the input document. Synchronous operations can use image files that are in JPEG or PNG format. Asynchronous operations also support PDF and TIFF format files.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `.*\S.*`

Required: No

Version

If the bucket has versioning enabled, you can specify the object version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `.*\S.*`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Warning

A warning about an issue that occurred during asynchronous text analysis (StartDocumentAnalysis (p. 217)) or asynchronous document text detection (StartDocumentTextDetection (p. 222)).

Contents

ErrorCode

The error code for the warning.

Type: String

Required: No

Pages

A list of the pages that the warning applies to.

Type: Array of integers

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Hard Limits in Amazon Textract

The following is a list of hard limits in Amazon Textract, which cannot be changed. For information about limitations in location and limits you can change see Amazon Textract Endpoints and Quotas. For information about limits you can change, see AWS Service Limits. To change a limit, see Create Case.

Amazon Textract

Limit	Description
Accepted File Formats	Synchronous operations support JPEG, PNG, and TIFF files. Asynchronous operations support JPEG, PNG, TIFF, and PDF (JPEG 2000-encoded images within PDFs are supported)..
File Size and Page Count Limits	For synchronous operations, JPEG, PNG, and TIFF files have a 10MB size limit. TIFF files also have a limit of 1 page. For asynchronous operations, JPEG and PNG files have a 10MB size limit. PDF and TIFF files have a 500MB limit. PDF and TIFF files have a limit of 3,000 pages.
PDF Specific Limits	The maximum height and width is 40 inches and 2880 points. PDFs cannot be password protected. PDFs can contain JPEG 2000 formatted images.
Document Rotation and Image Size	Amazon Textract supports all in-plane document rotations, for example 45 degree in-plane rotation. Amazon Textract supports images with a resolution less than or equal to 10000 pixels on all sides.
Text Alignment	Text can be text aligned horizontally within the document. Amazon Textract does not support vertical text alignment within the document.
Languages	Amazon Textract supports English, French, German, Italian, Portuguese and Spanish text detection. Amazon Textract will not return the language detected in its output.
Character Size	The minimum height for text to be detected is 15 pixels. At 150 DPI, this would be the same as 8 point font.
Character Type	Amazon Textract supports both handwritten and printed character recognition.
Characters	Amazon Textract detects the following characters: • a-z • A-Z • 0-9 • ä Ä ö Ö ü Ü ç Ç é É â Â ê Ê î Î ô Ô û Û à À è È ù Ù ë Ë ï Ï ü Ü á Á é É í Í ó Ó ú Ú ü Ü ñ Ñ ì Ì ò Ò ã Ã õ Õ • ! " # $ % ' & ( ) * + , - . / : ; = ? @ [\ ] ^ _ ` { | } ~ > < ° € £ ¥ ₹ ß ß ¿ ¡ € £ ¥ ₹ ø Ø œ Œ © ® ™ § ^{1} ^{2} ^{3} '

Document History for Amazon Textract

The following table describes important changes in each release of the *Amazon Textract Developer Guide*. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** May 29th, 2019

update-history-change	update-history-description	update-history-date
AnalyzeExpense Added (p. 256)	Amazon Textract now supports the analysis of invoice and receipt documents using the AnalyzeExpense API. This feature is only available in our Asia Pacific(Mumbai), Asia Pacific(Seoul), Asia Pacific(Singapore), Asia Pacific(Sydney), Canada(Central), Europe(Frankfurt), Europe(Ireland), Europe(London), U.S. East(N.Virginia), U.S. East(Ohio) U.S. West(N.California), and U.S West(Oregon) regions.	July 26, 2021
Augmented AI Support (p. 256)	Amazon Textract now supports Amazon Augmented AI for implementing human review.	December 3, 2019
New service and guide (p. 256)	Amazon Textract is now available for general use.	May 29, 2019
Support for selection elements (p. 256)	Amazon Textract can now detect selection elements (radio buttons and check boxes).	April 24, 2019
Release of Amazon Textract (p. 256)	This is the first release of the documentation for Amazon Textract.	November 28, 2018

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.