Amazon Textract

Developer Guide

Amazon Textract: Developer Guide

Table of Contents

What Is Amazon Textract?

Amazon Textract makes it easy to add document text detection and analysis to your applications. The Amazon Textract Text Detection API can detect text in a variety of documents including financial reports, medical records, 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. Amazon Textract includes simple, easy-to-use APIs that can analyze image files and PDF files. Amazon Textract is always learning from new data, and we're continually adding new features to the service.

The following are common use cases for using Amazon Textract:

- **Creating an intelligent search index** – Amazon Textract enables you to create libraries of text that is detected in image and PDF files.
- **Using intelligent text extraction for natural language processing (NLP)** – You can use Amazon Textract 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.
- **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 that's 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's document text detection. 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 Amazon Textract's 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.

An example is if you're building a mobile app to capture data from a single-page document that might require continued user engagement. If the uploaded image is too dark, the app might suggest photographing the image again with the camera flash turned on. A synchronous call would be appropriate because a user is waiting on the response. However, if you're building an application to index documents that have various numbers of pages, you might choose to send the same single-page form to the asynchronous call so that all of your documents can be processed in the same workflow.

By using AWS Batch, Amazon Textract is able to process multiple document images in a single operation. AWS Batch calls the Amazon Textract synchronous operations to process the document images. Note that PDF documents aren't supported.

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. 22)** – In this section, you set 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. 5)).

Amazon Textract provides operations for detecting text only and operations for analyzing text that find deeper relationships, such as form data and tables. For more information, see Detecting Text (p. 3) and Analyzing Text (p. 4).

Amazon Textract provides synchronous operations for processing small, single-page, documents and for getting near real-time responses. For more information, see Detecting and Analyzing Text in Single-Page Documents (p. 26). Amazon Textract also provides asynchronous operations that you can use to process larger, multipage documents. Asynchronous responses aren't in real time. For more information, see Detecting and Analyzing Text in Multipage Documents (p. 46).

When an Amazon Textract operation processes a document, the results are returned in an array of the section called "Block" (p. 118) objects. A `Block` object contains 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 Documents and Block Objects (p. 5). For examples that show how to use `Block` objects, see Examples (p. 71).

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. 118) 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. 8).

To detect text synchronously, use the DetectDocumentText (p. 97) 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 Detecting and Analyzing Text in Single-Page Documents (p. 26).

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. 114) to start processing an input document file. To get the results, call GetDocumentTextDetection (p. 106). The results are returned in one or more responses from `GetDocumentTextDetection`. For more information and an example, see Detecting and Analyzing Text in Multipage Documents (p. 46).

Analyzing Text

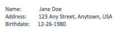
Amazon Textract analyzes documents and forms for relationships between detected text. Amazon Textract analysis operations return 3 categories of text extraction — text, forms, and tables.

Text Extraction

The raw text extracted from a document. For more information, see Lines and words of text (p. 8).

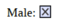
Form Extraction

Form data is linked 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. 9).



Name:	Jane Doe
Address:	123 Any Street, Anytown, USA
Birthdate:	12-26-1980

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

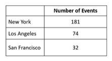


Male: ☒

For more information, see Selection elements (p. 13).

Table Extraction

Amazon Textract can extract tables, table cells, and the items within table cells.



	Number of Events
New York	181
Los Angeles	74
San Francisco	32

For more information, see Tables (p. 11). Selection elements can also be extracted from tables. For more information, see Selection elements (p. 13).

For analyzed items, Amazon Textract returns the following in multiple the section called "Block" (p. 118) 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. 93) 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. 38).

To detect text asynchronously, use StartDocumentAnalysis (p. 110) to start processing. To get the results, call GetDocumentAnalysis (p. 101). The results are returned in one or more responses from

`GetDocumentAnalysis`. For more information and an example, see Detecting and Analyzing Text in Multipage Documents (p. 46).

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 that's not related to the value of `FeatureTypes`).

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 and/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 or PDF format. Synchronous operations can process JPEG and PNG format images. Typically these are images of single-page documents that you've scanned. Asynchronous operations can also process documents that are in PDF format. Using PDF format files enables you to process multipage documents. For information about how Amazon Textract represents documents as `Block` objects, see Documents and Block Objects (p. 5).

For information about document limits, see Limits in Amazon Textract (p. 133).

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. 26). For asynchronous operations, you need to supply input documents in an Amazon S3 bucket. For more information, see Calling Amazon Textract Asynchronous Operations (p. 46).

Documents and Block Objects

When Amazon Textract processes a document, it creates a list of Block (p. 118) 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. 7)
- Lines and words of text (p. 8)
- Form data (Key-value pairs) (p. 9)
- Tables (p. 11)
- Selection elements (p. 13)

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 Text (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. 71).

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. 7).

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. 129) 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 Detecting and Analyzing Text in Single-Page Documents (p. 26).

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. 18).

Pages

A document consists of one or more pages. A the section called “Block” (p. 118) 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.

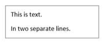
```
{
    "Geometry": ....
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "2602b0a6-20e3-4e6e-9e46-3be57fd0844b", Line - Hello, world.
                "82aedd57-187f-43dd-9eb1-4f312ca30042", Line - How are you?
                "52be1777-53f7-42f6-a7cf-6d09bdc15a30",
                "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 or PNG 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. Synchronous operations don't include the `Page` field, because they only process a scanned image (always a single-page document). PDF format documents aren't supported by synchronous operations.

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. 118) 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.

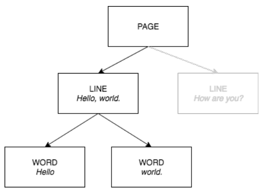
Detected text is returned in the `Text` field of a `Block` object. The `BlockType` field determines if the text is a line of text (LINE) or a word (WORD). A *WORD* is one or more ISO basic Latin script characters that aren't separated by spaces. A *LINE* is a string of tab-delimited and contiguous words.

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 “Documents and Block Objects” (p. 5).

To detect only lines and words, you can use DetectDocumentText (p. 97) or StartDocumentTextDetection (p. 114). 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. 93) or StartDocumentAnalysis (p. 110). For more information, see Analyzing Text (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,*:

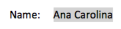
```
{
    "Geometry": {...},
    "Text": "Hello,",
    "BlockType": "WORD",
    "Confidence": 99.74746704101562,
    "Id": "7f97e2ca-063e-47a8-981c-8beee31afc01"
},
```

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

```
{
    "Geometry": {...},
    "Text": "world.",
    "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 partial document, Amazon Textract can identify a key (*Name:*) and a value (*Ana Carolina*).


Name: Ana Carolina

Detected key-value pairs are returned as Block (p. 118) objects in the responses from AnalyzeDocument (p. 93) and GetDocumentAnalysis (p. 101). 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. 71). For general information about how a document is represented by `Block` objects, see Documents and Block Objects (p. 5).

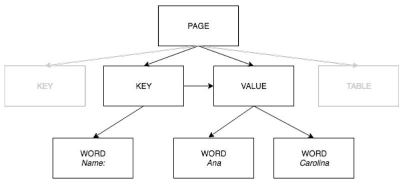
Block objects with the type KEY_VALUE_SET are the containers for KEY or VALUE Block objects that store information about linked text items detected in a document. You can use the `EntityType` attribute to determine if a block is a KEY or a VALUE.

- 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 that's 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 that's 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. 13).

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*).

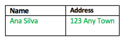
```
{
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "db553509-64ef-4ecf-ad3c-bea62cc1cd8a", // Ana
                "e5d7646c-eaa2-413a-95ad-f4ae19f53ef3"  // Carolina
            ]
        }
    ],
    "Confidence": 51.55965805053711,
    "Geometry": ....,
    "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:",
    "BlockType": "WORD",
    "Confidence": 99.56285858154297,
    "Id": "c734fca6-c4c4-415c-b6c1-30f7510b72ee"
},
 {
    "Geometry": {...},
    "Text": "Ana",
    "BlockType": "WORD",
    "Confidence": 99.52057647705078,
    "Id": "db553509-64ef-4ecf-ad3c-bea62cc1cd8a"
},
{
    "Geometry": {...},
    "Text": "Carolina",
    "BlockType": "WORD",
    "Confidence": 99.84207916259766,
    "Id": "e5d7646c-eaa2-413a-95ad-f4ae19f53ef3"
},
```

Tables

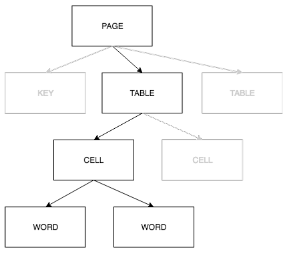
Amazon Textract can extract tables and the cells in a table. For example, in the following image, Amazon Textract detects a table with four cells.



Name	Address
Ana Silva	123 Any Town

Detected tables are returned as Block (p. 118) objects in the responses from AnalyzeDocument (p. 93) and GetDocumentAnalysis (p. 101). 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. 73). For general information about how a document is represented by `Block` objects, see Documents and Block Objects (p. 5).

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 Silva
                "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"
},
```

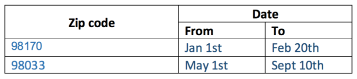
The Block type for the table cells is CELL. The `Block` object for each cell includes information about the cell location compared to other cells in the table. It also includes geometry information for the location of the cell on the document. In the preceding example, `505e9581-0d1c-42fb-a214-6ff736822e8c` 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. 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",
"BlockType": "WORD",
"Confidence": 99.81139373779297,
"Id": "e9108c8e-0167-4482-989e-8b6cd3c3653e"
},
```

Amazon Textract can provide information about the column and row spans that it detects in a table. For example, in the following table, the cell *Zip code* spans two rows. The cell *Date* spans two columns.

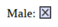


Zip code	Date	
	From	To
98170	Jan 1st	Feb 20th
98033	May 1st	Sept 10th

The cell *Block* stores column span and row span information in the `RowSpan` and `ColumnSpan` fields. In the previous example, the value of RowSpan for *Zipcode* is 2. The value of RowSpan for *From* is 1.

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. 9) and in tables (p. 11). For example, the following image is a check box that's detectable by Amazon Textract.



Detected selection elements are returned as Block (p. 118) objects in the responses from AnalyzeDocument (p. 93) and GetDocumentAnalysis (p. 101).

> **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 in 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. 118) objects.

For more information about key-value pairs, see Form Data (Key-Value Pairs) (p. 9).

The following JSON snippet shows the key for a key-value pair that contains a selection element. 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": "Void",
    "BlockType": "WORD",
    "Confidence": 54.06439208984375,
    "Id": "bd14cfd5-9005-498b-a7f3-45ceb171f0ff"
},
```

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

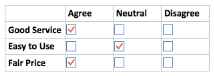
```
{
    "Relationships": [
        {
            "Type": "CHILD",
            "Ids": [
                "f2f5e8cd-e73a-4e99-a095-053acd3b6bfb"  //Selection element
            ]
        }
    ],
    "Confidence": 94.15619659423828,
    "Geometry": {
        "BoundingBox": {
            "Width": 0.017281491309404373,
            "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": "NOT_SELECTED",
    "Confidence": 74.14942932128906,
    "Id": "f2f5e8cd-e73a-4e99-a095-053acd3b6bfb"
}
```

Table Cells

Amazon Textract can detect selection elements inside a table cell.



	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. 11).

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's 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"
}
```

Item Location on a Document Page

Amazon Textract operations return the location and geometry of items found on a document page. DetectDocumentText (p. 97) and GetDocumentTextDetection (p. 106) return the location and geometry for lines and words, while AnalyzeDocument (p. 93) and GetDocumentAnalysis (p. 101) 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. 126)) information that's returned by the Amazon Textract operation in a Block (p. 118) object. The `Geometry` object contains two types of location and geometric information for detected items:

- An axis-aligned BoundingBox (p. 122) 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. 128) 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:",
    "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. 32). 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 width.
- 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
Bounding.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 example shows how to display a bounding box.

```
   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);

   }
```

Polygon

The polygon that's returned by `AnalyzeDocument` is an array of Point (p. 128) 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. 22).
2. Create an IAM User (p. 23).

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. 23)

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.

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 (Boto 3)
2. Create an access key for the user that you created in Create an IAM User (p. 23).

   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. 23).

   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.

   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.

   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. 25)

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. 38)

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.

Detecting and Analyzing Text in Single-Page Documents

Amazon Textract can detect and analyze text in single-page documents that are provided as images in JPEG or PNG format. The operations are synchronous and return results in near real time. For more information about documents, see Documents and Block Objects (p. 5).

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 format, see Detecting and Analyzing Text in Multipage Documents (p. 46).

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. 97) 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. 93) operation. For more information, see Analyzing Text (p. 4).

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. 123) input parameter. The document image can be in either PNG or JPEG format.

For a complete example, see Detecting Document Text with Amazon Textract (p. 32).

Request

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 that's 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 have to reference an image that's stored in an Amazon S3 bucket.

The following 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. 130) 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).

```
{
    "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": [
                        "d7fbd604-d609-4d69-857d-247a3f591238",
                        "b6c19a93-6493-4d8e-958f-853c8f7ca055"
                    ]
                }
            ],
            "BlockType": "PAGE",
            "Id": "56ec1d77-171f-4881-9852-2b5b7e761608"
        },
        {
            "Relationships": [
                {
                    "Type": "CHILD",
                    "Ids": [
                        "7f97e2ca-063e-47a8-981c-8beee31afc01",
                        "4b990aa0-af96-4369-b90f-dbe02538ed21"
                    ]
                }
            ],
            "Confidence": 99.63229370117188,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.2522801458835602,
                    "Top": 0.11217361688613892,
                    "Left": 0.16102784872055054,
                    "Height": 0.02862064726650715
                },
                "Polygon": [
                    {
                        "Y": 0.11217361688613892,
                        "X": 0.16102784872055054
                    },
                    {
                        "Y": 0.11217361688613892,
                        "X": 0.4133079946041107
                    },
                    {
                        "Y": 0.14079426229000092,
                        "X": 0.4133079946041107
                    },
                    {
                        "Y": 0.14079426229000092,
                        "X": 0.16102784872055054
                    }
                ]
            },
            "Text": "Hello, world.",
            "BlockType": "LINE",
            "Id": "d7fbd604-d609-4d69-857d-247a3f591238"
        },
        {
            "Relationships": [
                {
                    "Type": "CHILD",
                    "Ids": [
                        "5a87995f-5ba0-4010-a2b1-ac1f100a6412",
                        "f287dfbb-94fb-439c-9907-a431f9e2269e",
                        "74cf9c3a-6aa6-427b-9e0c-f29741453e5f"
                    ]
                }
            ],
            "Confidence": 99.64396667480469,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.26536890864372253,
                    "Top": 0.1513642966747284,
                    "Left": 0.1621238887310028,
                    "Height": 0.02884846366941929
                },
                "Polygon": [
                    {
                        "Y": 0.1513642966747284,
                        "X": 0.1621238887310028
                    },
                    {
                        "Y": 0.1513642966747284,
                        "X": 0.42749279737472534
                    },
                    {
                        "Y": 0.18021276593208313,
                        "X": 0.42749279737472534
                    },
                    {
                        "Y": 0.18021276593208313,
                        "X": 0.1621238887310028
                    }
                ]
            },
            "Text": "How are you?",
            "BlockType": "LINE",
            "Id": "b6c19a93-6493-4d8e-958f-853c8f7ca055"
        },
        {
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.11247961968183517,
                        "Top": 0.11266519874334335,
                        "Left": 0.16102784872055054,
                        "Height": 0.02812906540930271
                    },
                    "Polygon": [
                        {
                            "Y": 0.11266519874334335,
                            "X": 0.16102784872055054
                        },
                        {
                            "Y": 0.11266519874334335,
                            "X": 0.2735074758529663
                        },
                        {
                            "Y": 0.14079426229000092,
                            "X": 0.2735074758529663
                        },
                        {
                            "Y": 0.14079426229000092,
                            "X": 0.16102784872055054
                        }
                    ]
                },
                "Text": "Hello,",
                "BlockType": "WORD",
                "Confidence": 99.74746704101562,
                "Id": "7f97e2ca-063e-47a8-981c-8beee31afc01"
            },
            {
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.12675164639949799,
                        "Top": 0.11217361688613892,
                        "Left": 0.28655633330345154,
                        "Height": 0.027762649580836296
                    },
                    "Polygon": [
                        {
                            "Y": 0.11217361688613892,
                            "X": 0.28655633330345154
                        },
                        {
                            "Y": 0.11217361688613892,
                            "X": 0.4133079946041107
                        },
                        {
                            "Y": 0.13993626832962036,
                            "X": 0.4133079946041107
                        },
                        {
                            "Y": 0.13993626832962036,
                            "X": 0.28655633330345154
                        }
                    ]
                },
                "Text": "world.",
                "BlockType": "WORD",
                "Confidence": 99.5171127319336,
                "Id": "4b990aa0-af96-4369-b90f-dbe02538ed21"
            },
            {
                "Geometry": {
                    "BoundingBox": {
```

```
                    "Width": 0.09069254994392395,
                    "Top": 0.15190091729164124,
                    "Left": 0.1621238887310028,
                    "Height": 0.02474799193441868
                },
                "Polygon": [
                    {
                        "Y": 0.15190091729164124,
                        "X": 0.1621238887310028
                    },
                    {
                        "Y": 0.15190091729164124,
                        "X": 0.25281643867492676
                    },
                    {
                        "Y": 0.17664891481399536,
                        "X": 0.25281643867492676
                    },
                    {
                        "Y": 0.17664891481399536,
                        "X": 0.1621238887310028
                    }
                ]
            },
            "Text": "How",
            "BlockType": "WORD",
            "Confidence": 99.76000213623047,
            "Id": "f287dfbb-94fb-439c-9907-a431f9e2269e"
        },
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.06475766748189926,
                    "Top": 0.15600404143333435,
                    "Left": 0.2611062824726105,
                    "Height": 0.020542677491903305
                },
                "Polygon": [
                    {
                        "Y": 0.15600404143333435,
                        "X": 0.2611062824726105
                    },
                    {
                        "Y": 0.15600404143333435,
                        "X": 0.32586392760276794
                    },
                    {
                        "Y": 0.17654670774936676,
                        "X": 0.32586392760276794
                    },
                    {
                        "Y": 0.17654670774936676,
                        "X": 0.2611062824726105
                    }
                ]
            },
            "Text": "are",
            "BlockType": "WORD",
            "Confidence": 99.70841217041016,
            "Id": "5a87995f-5ba0-4010-a2b1-ac1f100a6412"
        },
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.092494897544384,
                    "Top": 0.1513642966747284,
                        "Left": 0.33499792218208313,
                        "Height": 0.02884846366941929
                    },
                    "Polygon": [
                        {
                            "Y": 0.1513642966747284,
                            "X": 0.33499792218208313
                        },
                        {
                            "Y": 0.1513642966747284,
                            "X": 0.42749279737472534
                        },
                        {
                            "Y": 0.18021276593208313,
                            "X": 0.42749279737472534
                        },
                        {
                            "Y": 0.18021276593208313,
                            "X": 0.33499792218208313
                        }
                    ]
                },
                "Text": "you?",
                "BlockType": "WORD",
                "Confidence": 99.46348571777344,
                "Id": "74cf9c3a-6aa6-427b-9e0c-f29741453e5f"
            }
        ],
        "DocumentMetadata": {
            "Pages": 1
        }
}
```

Detecting Document Text with Amazon Textract

To detect text in a document, you use the DetectDocumentText (p. 97) 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. 23).

   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. 23).

2. Upload a document to your S3 bucket.

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

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

if __name__ == "__main__":

    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()
```

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. 93) operation, and pass a document file as input. `AnalyzeDocument` returns a JSON structure that contains the analyzed text. For more information, see Analyzing Text (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. 23).

   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. 23).

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 Console 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.

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()

if __name__ == "__main__":

    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"])
    #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()
```

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.

Detecting and Analyzing Text in Multipage Documents

Amazon Textract can detect and analyze text in multipage documents that are in PDF format. 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 format. Single-page documents processed with asynchronous operations can be in JPEG, PNG, 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. 114) and GetDocumentTextDetection (p. 106). 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. 110) and GetDocumentAnalysis (p. 101). For more information, see Analyzing Text (p. 4).

Topics

Calling Amazon Textract Asynchronous Operations

Amazon Textract provides an asynchronous API that you can use to process multipage documents in PDF format. You can also use asynchronous operations to process single-page documents that are in JPEG, PNG, 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 the section called “StartDocumentAnalysis” (p. 110), and the section called “GetDocumentAnalysis” (p. 101).

For an example, see Detecting or Analyzing Text in a Multipage Document (p. 57).

Amazon Textract asynchronously processes a document that's stored in an Amazon S3 bucket. You start processing by calling a `Start` operation, such as StartDocumentTextDetection (p. 114). 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. 106), to get the results of the request.

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 that's stored in an Amazon S3 bucket. In the diagram, an Amazon SQS queue gets the completion status from the Amazon SNS topic.

The process is the same for analyzing text. You start analyzing text by calling the section called “StartDocumentAnalysis” (p. 110). You get the results by calling the section called “GetDocumentAnalysis” (p. 101).

Starting Text Detection

You start an Amazon Textract text detection request by calling StartDocumentTextDetection (p. 114). 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/roleopic"
    },
    "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. 55).

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.

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. 69).

To get the status information that's 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 that's published by Amazon Textract when a text detection request completes. For more information, see Detecting or Analyzing Text in a Multipage Document (p. 57). 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, a maximum of 1,000 results is returned. If the operation doesn't return the entire set of results, a pagination token for the next page is returned in the operation response. If you have a pagination token from a previous Get request, use it with `NextToken` to get the next page of results.

Note
Amazon Textract retains the results of asynchronous operations for 7 days. You won't be able to 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 available from `DocumentMetadata`. The detected text is returned in the `Blocks` array. For information about `Block` objects, see Documents and Block Objects (p. 5).

```
{
    "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": "elloworio",
            "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": 0.29955607652664185,
                "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": 0.23131252825260162
                },
                "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": "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 for use with an Amazon SNS topic and an Amazon SQS queue.

Note
If you're using these instructions to set up the Detecting or Analyzing Text in a Multipage Document (p. 57) example, you don't need to do steps 3, 4, 5, and 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. 22).

   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. 23).
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.
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. 56).

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": "arn: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. 57).

Giving Amazon Textract Access to Your Amazon SNS Topic

Amazon Textract needs permission to send the completion status of an asynchronous operation 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 **EC2**.
6. Choose **Next: Permissions**.
7. For **Attach permissions policies**, select the check box next to the policy *AmazonTextractServiceRole*. To display the policy in the list, enter part of the policy name in the **Filter policies** query filter.
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 in **Role description**, 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 update the trust policy 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. 46).

The procedure enables you to choose the type of processing that you want the code to do—text detection or text analysis. If you choose text detection, StartDocumentTextDetection (p. 114) is called to start text detection. The results are returned by calling GetDocumentTextDetection (p. 106). If you choose text analysis, StartDocumentAnalysis (p. 110) is called to start text analysis. You get the results by calling GetDocumentAnalysis (p. 101).

The processing results are returned in an array of the section called “Block” (p. 118) objects. They are different depending on the type of processing. For information about text detection blocks, see Detecting Text (p. 3). For text analysis blocks, see Analyzing Text (p. 4).

The example code in the procedure shows you how to do the following steps:

1. Create the Amazon SNS topic and the Amazon SQS queue.
2. Subscribe the Amazon SQS queue to the Amazon SNS topic.
3. Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue.
4. Start processing the document. Start text detection by calling StartDocumentTextDetection (p. 114). Start text analysis by calling StartDocumentAnalysis (p. 110).
5. Get the completion status from the Amazon SQS queue. The example 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 an important consideration if other applications are using the same queue and topic. For simplicity, the example deletes jobs that don't match. Consider adding them to an Amazon SQS dead-letter queue for further investigation.
6. Get and display the processing results by calling GetDocumentTextDetection (p. 106) or GetDocumentAnalysis (p. 101).
7. Delete the Amazon SNS topic and the Amazon SQS queue.

Detecting or Analyzing Text

The example code for this procedure is provided in Java and Python. You need to have the appropriate AWS SDK installed. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 23).

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. 55). You

don't need to do steps 3, 4, 5, and 6 because the example code creates and configures the Amazon SNS topic and Amazon SQS queue.

2. Upload a multipage document file in PDF format to your Amazon S3 bucket. (Single-page documents in JPEG, PNG, or PDF format can also be processed).

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

   - Replace the value of `roleArn` with the IAM role ARN that you saved in Giving Amazon Textract Access to Your Amazon SNS Topic (p. 56).
   - 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.

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("Video 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();
   }
}
```

Python

```
import boto3
import json
import sys
import time

class ProcessType:
    DETECTION = 1
    ANALYSIS = 2


class DocumentProcessor:
    jobId = ''
    textract = boto3.client('textract')
    sqs = boto3.client('sqs')
    sns = boto3.client('sns')

    roleArn = ''
    bucket = ''
    document = ''
    sqsQueueUrl = ''
    snsTopicArn = ''
    processType = ''

    def main(self):
        self.roleArn = 'role'
        self.bucket = 'bucket'
        self.document = 'document'

        self.CreateTopicandQueue()
        self.ProcessDocument(ProcessType.DETECTION)
        self.DeleteTopicandQueue()


    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


if __name__ == "__main__":
    analyzer=DocumentProcessor()
    analyzer.main()
```

4. Build and 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. 46). For an example, see Detecting or Analyzing Text in a Multipage Document (p. 57).

The payload is in the following JSON format:

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

Name	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. 114).
Status	The status of the job. Valid values are SUCCEEDED, FAILED, or ERROR.
API	The Amazon Textract operation used to analyze the input document.
JobTag	The user-specified identifier for the job. You specify `JobTag` in a call to the `Start` operation, such as StartDocumentTextDetection (p. 114).
Timestamp	The Unix timestamp for when the job finished.
DocumentLocation	Details about the document that was processed. Includes the file name and the Amazon S3 bucket that the file is stored in.

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

- Ensure that your document text is in a language that Amazon Textract supports. Currently, Amazon Textract only supports English.
- 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, JPEG, and PNG), don't convert or downsample the document before uploading it to Amazon Textract.

Amazon Textract table extraction works best under the following conditions.

- The 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.
- The text within the table is upright. For example, the text isn't rotated relative to other text on the page.

You might see inconsistent results with the following conditions. We recommend using text detection (p. 3) as a workaround.

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

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 enables you to make informed decisions on how you want to use the results.

You should enforce a minimum confidence score threshold in applications that are sensitive to detection errors (false positives). The application should discard results below that threshold or apply a higher level of human scrutiny. The optimal threshold depends on the application. For archival purposes 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. 118) 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. 93). 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. 110), you need to change the example code to accommodate multiple batches of returned `Block` objects.

For examples that show you other ways to use Amazon Textract, see Other Examples (p. 76).

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. 23).
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. 23).

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. 118) objects that are stored in a map. Block objects are returned from a call to the section called “AnalyzeDocument” (p. 93). For more information, see Form Data (Key-Value Pairs) (p. 9).

The functions that are specific to Amazon Textract are:

- `get_kv_map` – Calls AnalyzeDocument (p. 93), 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.

Note
You can download the source file from textract-python-kv-parser.py.

To extract key-value pairs from a form document

1. Configure your environment. For more information, see Prerequisites (p. 71).
2. Save the following example code to a file named *textract-python-kv-parser.py*.

```
import boto3
import sys
import re
import json

file_name = sys.argv[1]

# get the results
client = boto3.client(
         service_name='textract',
         region_name= 'us-east-1',
         endpoint_url='https://textract.us-east-1.amazonaws.com',
)


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
    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


# MAIN PROGRAM
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))
```

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 part of the input document. If the key is detected, the program displays the value that's associated with the key.

Exporting Tables into a CSV File

This Python example shows how to export tables into a comma-separated values (CSV) file. Table information is returned as the section called “Block” (p. 118) objects from a call to the section called “AnalyzeDocument” (p. 93). For more information, see Tables (p. 11). The `Block` objects are stored in a map structure that's used to export the table data into a CSV file.

The functions that are specific to Amazon Textract are:

- `get_table_csv_results` – Calls AnalyzeDocument (p. 93), 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.

Note
You can download the source code from textract-python-table-parser.py.

To export tables into a CSV file

1. Configure your environment. For more information, see Prerequisites (p. 71).
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



file_name = sys.argv[1]

# get the results
client = boto3.client(
         service_name='textract',
         region_name= 'us-east-1',
         endpoint_url='https://textract.us-east-1.amazonaws.com',
)




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
    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


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)
```

3. At the command prompt, enter the following command. Replace `file` with 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 to a file named `output.csv`.

Other Examples

These are links to other code examples that you can use with Amazon Textract.

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. 118) objects returned by Amazon Textract operations.
Amazon Textract Documentation Code Examples	Code examples used in this section.
Textractor	Shows how to convert Amazon Textract output into multiple formats.

Amazon Textract Security

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

Authentication and Access Control for Amazon Textract

Access to Amazon Textract requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access AWS resources, such as an Amazon S3 bucket or an AWS Lambda function. An asynchronous job is the only Amazon Textract resource, but it doesn't have an associated Amazon Resource Name (ARN). The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon Textract to help secure access to Amazon Textract.

Authentication

You can access AWS as any of the following types of identities:

- **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 user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create an asynchronous job in Amazon Textract). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several

SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. Amazon Textract supports *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*.

- **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session. IAM roles with temporary credentials are useful in the following situations:

  - **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*.

  - **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

  - **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*.

Access Control

You can have valid credentials to authenticate your requests, but if you don't have permissions, you can't create or access Amazon Textract resources. For example, you must have permissions to create an Amazon Textract asynchronous job.

The following sections describe how to manage permissions for Amazon Textract. We recommend that you read the overview first.

- Overview of Managing Access Permissions to Your Amazon Textract Resources (p. 79)
- Using Identity-based Policies (IAM Policies) for Amazon Textract (p. 82)
- Amazon Textract API Permissions: Actions, Permissions, and Resources Reference (p. 84)

Overview of Managing Access Permissions to Your Amazon Textract Resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles). Some services (such as AWS Lambda) also support attaching permissions policies to resources.

> **Note**
> An *account administrator* (or administrator user) is a user with administrator permissions. For more information, see IAM Best Practices in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

Amazon Textract Resources and Operations

In Amazon Textract, the primary resource is an *asynchronous job*. As mentioned previously, asynchronous job don't have Amazon Resource Names (ARNs) associated with them. However, some other services that you use with Amazon Textract do have unique ARNs associated with their resources. For example, you can use Amazon S3 buckets to store documents that are used by the Amazon Textract API. The following table shows the ARNs for an Amazon S3 bucket and the key that's used to identify the stored object (document image file) in the bucket.

Resource Type	ARN Format
Amazon S3 bucket ARN	`arn:aws:s3:::`*`bucket_name`*
Amazon S3 object ARN	`arn:aws:s3:::`*`bucket_name/key_name`*

Amazon Textract provides a set of operations to work with documents. For a list of available operations, see Amazon Textract API Reference (p. 92).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account or an IAM user) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create an asynchronous job, your AWS account is the owner of the resource (in Amazon Textract, the resource is an asynchronous job).
- If you create an IAM user in your AWS account and grant permissions to create an asynchronous job to that user, the user can create an asynchronous job. However, your AWS account, to which the user belongs, owns the asynchronous job resource.

Managing Access to Resources

A *permissions policy* describes who has access to which resources. The following section explains the available options for creating permissions policies.

> **Note**
> This section discusses using IAM in the context of Amazon Textract. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM policies). Policies attached to a resource are referred to as *resource-based* policies. Amazon Textract supports identity-based policies.

Topics

Identity-based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create an Amazon Textract resource, such as an asynchronous job, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in account A can create a role to grant cross-account permissions to another AWS account (for example, account B) or an AWS service as follows:
  1. The account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in account A.
  2. The account A administrator attaches a trust policy to the role that identifies account B as the principal who can assume the role.
  3. The account B administrator can then delegate permissions to assume the role to any users in account B. Doing this allows users in account B to create or access resources in account A. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following is an example policy that allows access to the DetectDocumentText (p. 97) operation.

```
"Version": "2012-10-17",
             "Statement": [
        {
            "Sid": "AllowsTextDetection",
            "Effect": "Allow",
            "Action": [
                "textract:DetectDocumentText"
            ],
            "Resource": "*"
        }
    ]
}
```

For more information about using identity-based policies with Amazon Textract, see Using Identity-based Policies (IAM Policies) for Amazon Textract (p. 82). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-based Policies

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Amazon Textract doesn't support resource-based policies.

To access images stored in an Amazon S3 bucket, you must have permission to access objects in the S3 bucket. With this permission, Amazon Textract can download images from the S3 bucket. The following example policy allows the user to perform the `s3:GetObject` action on the S3 bucket named Tests3bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::Tests3bucket"
            ]
        }
    ]
}
```

To use an S3 bucket with versioning enabled, add the `s3:GetObjectVersion` action, as shown in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::Tests3bucket"
            ]
        }
    ]
```

Specifying Policy Elements: Actions, Effects, and Principals

For each Amazon Textract resource, the service defines a set of API operations. To grant permissions for these API operations, Amazon Textract defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see Amazon Textract Resources and Operations (p. 79) and Amazon Textract API Permissions: Actions, Permissions, and Resources Reference (p. 84).

The following are the most basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. For more information, see Amazon Textract Resources and Operations (p. 79).

- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, you can use `ListCollections` to list collections.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user can't access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only). Amazon Textract doesn't support resource-based policies.

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a list showing all of the Amazon Textract API operations and the resources that they apply to, see Amazon Textract API Permissions: Actions, Permissions, and Resources Reference (p. 84).

Specifying Conditions in a Policy

When you grant permissions, you can use the access-policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

To express conditions, you use predefined condition keys. There are no condition keys specific to Amazon Textract. However, there are AWS-wide condition keys that you can use as appropriate. For a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*.

Using Identity-based Policies (IAM Policies) for Amazon Textract

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and then grant permissions to perform operations on Amazon Textract resources.

Important
We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your Amazon Textract resources. For more information, see Overview of Managing Access Permissions to Your Amazon Textract Resources (p. 79).

Topics

The following shows an example of a permissions policy.

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "textract:DetectDocumentText",
                "textract:AnalyzeDocument"
            ],
            "Resource": "*"
        }
    ]
```

This policy example grants a user access to the synchronous Amazon Textract operations.

For a table showing all of the Amazon Textract API operations and the resources that they apply to, see Amazon Textract API Permissions: Actions, Permissions, and Resources Reference (p. 84).

Permissions Required to Use the Amazon Textract Console

Amazon Textract doesn't require any additional permissions when you're working with the Amazon Textract console.

AWS Managed (Predefined) Policies for Amazon Textract

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Textract:

- **AmazonTextractFullAccess** – Grants full access to Amazon Textract.
- **AmazonTextractServiceRole** – Allows Amazon Textract to call Amazon SNS services on your behalf.

Note
You can review these permissions policies by signing in to the IAM console and searching for specific policies there.
These policies work when you're using AWS SDKs or the AWS CLI.

You can also create your own custom IAM policies to allow permissions for Amazon Textract actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

Customer Managed Policy Examples

In this section, you can find example user policies that grant permissions for various Amazon Textract actions. These policies work when you're using AWS SDKs or the AWS CLI. When you're using the console, you need to grant additional permissions that are specific to the console. For more information, see Permissions Required to Use the Amazon Textract Console (p. 83).

Examples

- Example 1: Allow a User Access to Only Asynchronous Amazon Textract Operations (p. 83)
- Example 2: Allow a User Full Access to Amazon Textract Operations (p. 84)

Example 1: Allow a User Access to Only Asynchronous Amazon Textract Operations

The following example grants access to asynchronous Amazon Textract operations.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "textract:StartDocumentTextDetection",
                "textract:StartDocumentAnalysis",
                "textract:GetDocumentTextDetection",
                "textract:GetDocumentAnalysis"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 2: Allow a User Full Access to Amazon Textract Operations

The following example grants full access to Amazon Textract operations.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "textract:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Textract API Permissions: Actions, Permissions, and Resources Reference

When you're setting up access control and writing a permissions policy that you can attach to an IAM identity (identity-based policies), you can use the following list as a reference (for more information, see Access Control (p. 78)). The table list includes each Amazon Textract API operation, the corresponding actions for which you can grant permissions to perform the action, and the AWS resource that you can grant the permissions for. You specify the actions in the policy's `Action` field, and you specify the resource value in the policy's `Resource` field.

You can use AWS-wide condition keys in your Amazon Textract policies to express conditions. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

Note
To specify an action, use the `textract` prefix followed by the API operation name (for example, `textract:DetectDocumentText`).

Amazon Textract API and Required Permissions for Actions

API Operation: DetectDocumentText

Required Permissions (API Action): hieroglyph:DetectDocumentText

None used.

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 `ResponseTime` and `Data Samples` 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. 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"
        }

    ]
}
```

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. 118) 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. 110).

For more information, see Document Text Analysis.

Request Syntax

```
{
   "Document": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   },
   "FeatureTypes": [ "string" ]
}
```

Request Parameters

The request accepts the following data in JSON format.

Document (p. 93)

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. 123) object

Required: Yes

FeatureTypes (p. 93)

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

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"
      }
   ],
   "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. 94)

The items that are detected and analyzed by `AnalyzeDocument`.

Type: Array of Block (p. 118) objects

DocumentMetadata (p. 94)

Metadata about the analyzed document. An example is the number of pages.

Type: DocumentMetadata (p. 125) object

Errors

AccessDeniedException

You aren't authorized to perform the action.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document.

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 5 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.

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. Documents for asynchronous operations can also 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 118) 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. 114).

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. 97)

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. 123) 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"
      }
   ],
   "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. 97)

An array of `Block` objects that contain the text that's detected in the document.

Type: Array of Block (p. 118) objects

DocumentMetadata (p. 97)

Metadata about the document. It contains the number of pages that are detected in the document.

Type: DocumentMetadata (p. 125) object

Errors

AccessDeniedException

You aren't authorized to perform the action.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document.

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 5 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.

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. Documents for asynchronous operations can also 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3

- AWS SDK for Python
- AWS SDK for Ruby V2

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. 110), 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. 118) objects. The following types of information are returned:

- Form data (key-value pairs). The related information is returned in two Block (p. 118) 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. 101)

A unique identifier for the text-detection job. The `JobId` is returned from `StartDocumentAnalysis`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: Yes

MaxResults (p. 101)

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. 101)

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"
      }
   ],
   "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. 102)

The results of the text-analysis operation.

Type: Array of Block (p. 118) objects

DocumentMetadata (p. 102)

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. 125) object

JobStatus (p. 102)

The current status of the text detection job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED | PARTIAL_SUCCESS`

NextToken (p. 102)

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. 102)

The current status of an asynchronous document-analysis operation.

Type: String

Warnings (p. 102)

A list of warnings that occurred during the document-analysis operation.

Type: Array of Warning (p. 132) objects

Errors

AccessDeniedException

You aren't authorized to perform the action.

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. 101) or to GetDocumentAnalysis (p. 101).

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

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 Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 114), 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. 118) 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. 106)

A unique identifier for the text detection job. The `JobId` is returned from `StartDocumentTextDetection`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: Yes

MaxResults (p. 106)

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. 106)

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"
      }
   ],
   "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. 107)

The results of the text-detection operation.

Type: Array of Block (p. 118) objects

DocumentMetadata (p. 107)

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. 125) object

JobStatus (p. 107)

The current status of the text detection job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED | PARTIAL_SUCCESS`

NextToken (p. 107)

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. 107)

The current status of an asynchronous text-detection operation for the document.

Type: String

Warnings (p. 107)

A list of warnings that occurred during the text-detection operation for the document.

Type: Array of Warning (p. 132) objects

Errors

AccessDeniedException

You aren't authorized to perform the action.

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. 101) or to GetDocumentAnalysis (p. 101).

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

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 Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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, and PDF format. The documents are stored in an Amazon S3 bucket. Use DocumentLocation (p. 124) 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. 101), 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",
   "NotificationChannel": {
      "RoleArn": "string",
      "SNSTopicArn": "string"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientRequestToken (p. 110)

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.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: No

DocumentLocation (p. 110)

The location of the document to be processed.

Type: DocumentLocation (p. 124) object

Required: Yes

FeatureTypes (p. 110)

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. 110)

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

NotificationChannel (p. 110)

The Amazon SNS topic ARN that you want Amazon Textract to publish the completion status of the operation to.

Type: NotificationChannel (p. 127) 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. 111)

The identifier for the document text detection job. Use `JobId` to identify the job in a subsequent call to `GetDocumentAnalysis`.

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.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document.

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 5 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

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.

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. Documents for asynchronous operations can also 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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, and PDF format. The documents are stored in an Amazon S3 bucket. Use DocumentLocation (p. 124) 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. 106), 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",
   "NotificationChannel": {
      "RoleArn": "string",
      "SNSTopicArn": "string"
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

ClientRequestToken (p. 114)

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.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-_]+$`

Required: No

DocumentLocation (p. 114)

The location of the document to be processed.

Type: DocumentLocation (p. 124) object

Required: Yes

JobTag (p. 114)

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

NotificationChannel (p. 114)

The Amazon SNS topic ARN that you want Amazon Textract to publish the completion status of the operation to.

Type: NotificationChannel (p. 127) 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. 115)

The identifier of the text detection job for the document. Use `JobId` to identify the job in a subsequent call to `GetDocumentTextDetection`.

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.

HTTP Status Code: 400

BadDocumentException

Amazon Textract isn't able to read the document.

HTTP Status Code: 400

DocumentTooLargeException

The document can't be processed because it's too large. The maximum document size for synchronous operations 5 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

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.

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. Documents for asynchronous operations can also 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 97)), you get information about the detected words and lines of text. In text analysis (for example AnalyzeDocument (p. 93)), 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. 97), the array of `Block` objects is the entire set of results. In asynchronous operations, such as GetDocumentAnalysis (p. 101), 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. `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. 126) 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 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. 129) 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 spans. `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

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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

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 5 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 5242880.

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 5 MB.

Type: S3Object (p. 130) 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

DocumentLocation

The Amazon S3 bucket that contains the document to be processed. It's used by asynchronous operations such as StartDocumentTextDetection (p. 114).

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. 130) 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

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.

Contents

BoundingBox

An axis-aligned coarse representation of the location of the recognized item on the document page.

Type: BoundingBox (p. 122) object

Required: No

Polygon

Within the bounding box, a fine-grained polygon around the recognized item.

Type: Array of Point (p. 128) 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

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. 114).

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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

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. 126) object that's returned in a Block (p. 118) 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

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. 118).

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`.

Type: String

Valid Values: `VALUE | CHILD`

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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

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.

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 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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

Warning

A warning about an issue that occurred during asynchronous text analysis (StartDocumentAnalysis (p. 110)) or asynchronous document text detection (StartDocumentTextDetection (p. 114)).

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 Go - Pilot
- AWS SDK for Java
- AWS SDK for Ruby V2

Limits in Amazon Textract

The following is a list of limits in Amazon Textract that you can't change.

Amazon Textract

- The maximum document image (JPEG/PNG) size is 5 MB.
- The maximum PDF file size is 500 MB.
- The maximum number of pages in a PDF file is 3000.
- The minimum height for text to be detected is 15 pixels. At 150 DPI, this would be equivalent to 8-pt font.
- Documents can be rotated a maximum of +/- 10% from the vertical axis. Text can be text aligned horizontally within the document.
- Amazon Textract doesn't support the detection of handwriting.
- Amazon Textract synchronous operations (`DetectDocumentText` and `AnalyzeDocument`) support the PNG and JPEG image formats. Asynchronous operations (`StartDocumentTextDetection`, `StartDocumentAnalysis`) also support the PDF file format.

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
New service and guide (p. 134)	Amazon Textract is now available for general use.	May 29, 2019
Support for selection elements (p. 134)	Amazon Textract can now detect selection elements (radio buttons and check boxes).	April 24, 2019
Release of Amazon Textract (p. 134)	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*.