TWAIN Direct™ Specification

(DRAFT COPY)

April 21st 2015
Revision 0.8

This is a draft copy of the proposed TWAIN Direct Specification. It's contents may be added to, changed or deleted at any time.

History

Date	Version	Comment
July 1st, 2014	0.01	Initial draft.
December 4th, 2014	0.02	Revision after comments (folded in the contents of the Introduction to TWAIN Direct)
February 3rd, 2015	0.03	Revisions made as part of the February Tech meeting
February 10th, 2015	0.04	Added Rebecca's comments, finish resolving all comments
March 3rd, 2015	0.05	Added new metadata content
March 17th, 2015	0.06	Comments from Mihail, metadata changes
March 21st, 2015	0.06	Expanded the section on Exceptions
April 21st, 2015	0.07	Prep work for releasing draft copy

Notes

Notes
• (none)

Acknowledgements

Contents

Glossary of Terms

This section establishes the meaning of words used within the Specification.

Word	Meaning
action	A TWAIN Direct command (e.g. “configure”, “scan”).
application	A program that sends TWAIN Direct commands to a scanner.
attribute	A configurable item, such as compression, autocrop, autodeskew, etc.
communication manager	A system that discovers scanners, registers them and provides cloud and/or local area net communication channels.
exception	A TWAIN Direct directive that changes the way a TWAIN Direct task is evaluated by a scanner, when it cannot exactly match a specific request within a task.
JSON	A lightweight data-interchange format (see www.json.org).
pixelFormat	The combination of a colorspace and a bit depth, for instance, rgb24 indicates a color image with 24 bits of depth.
scanner	Any device that captures images for an application.
source	A physical provider of images, such as a flatbed or an automatic document feeder.
stream	A collection of one or more sources, which combined together results in a stream of images during scanning.
task	A TWAIN Direct construct used to issue actions to a scanner.
user	A person in control of an application and a scanner.

References

This section lists standards and guides that are cited in this document.

Word	Meaning
Google JSON Style Guide	Google JSON Style Guide https://google-styleguide.googlecode.com/svn/trunk/jsoncstyleguide.xml
JSON	ECMA-404 The JSON Data Interchange Standard http://json.org
PDF/raster	PDF Raster Documents http://pdfraster.org
UUID	A Universally Unique IDentifier (UUID) URN Namespace http://www.ietf.org/rfc/rfc4122.txt

Overview

Summary

TWAIN Direct is a command set used by applications to control scanners. It has two main goals: to support direct communication between an application and a scanner, and to do so in a way that minimizes the effort needed by application developers..

Target Audience

This document is meant to be read by both application developers and scanner vendors.

A scanning environment consists of the following:

- A scanner (and therefore a scanner vendor).
- An end user, who operates the scanner.
- Application software used by the end user (and therefore an application vendor).
- IT departments to maintain the environment for the application and the scanner.

TWAIN Direct is primarily focused on application vendors. Applications represent the needs of the end users, and there are considerably more application writers than scanner vendors. Improving the development experience in this area offers the most potential for influencing the market. The expectation is that focusing on this segment will inevitably lead to benefits for the other segments.

Current Challenges for Application Vendors

In today's environment, when a developer decides to add scanning capabilities to their application, they must be prepared to encounter the following issues:

- For historical reasons Image Capture APIs are generally limited to or best supported by the C and C++ programming languages. Third party toolkits are required when interfacing to other popular languages or web browsers.

- Developers must decide which Image Capture API to use (e.g., TWAIN, WIA, ISIS, SANE, etc). This decision is influenced by the operating system the application runs on and the APIs supported by the scanner. This makes it harder for developers to switch to different scanner products or to support new ones.

- Image Capture APIs are complex, reflecting the native complexity of scanners, which

often support a large number of possible settings (e.g., compression, brightness, contrast, region of interest, deskew, barcode detection, etc). The TWAIN Specification currently details more than 150 capabilities. Scanner vendors may add their own custom capabilities. This complexity often causes developers to seek a minimal solution to their problems, even if the scanner is capable of producing a better user experience.

- Instead of focusing solely on solving the end user's problem, the bulk of development time is devoted to defensive programming dealing with the variable feature set of scanners and the reliability of their drivers. Development of a robust application may take weeks or months.

- Scanner vendors supply drivers for their devices (e.g., TWAIN, WIA, ISIS, SANE, etc). End users and IT personnel must install these drivers on their systems and upgrade them as needed. In this model, applications talk to drivers, and drivers talk to scanners. The communication between a driver and a scanner is proprietary, making it harder for developers to diagnose problems without the aid of the scanner vendor.

- The current Image Capture APIs were developed for peripheral communication protocols (e.g., IEEE 1284, SCSI, USB, etc). Application control of network scanners has not been standardized and adopted across the industry for all operating system platforms.

TWAIN Direct

What is TWAIN Direct?

- It is a command set that allows applications to talk directly to scanners without vendor specific drivers (referred to as Scanning Without Requiring Drivers, or SWORD).

- It is an initiative to minimize the coding effort developers need to support fully featured image capture solutions in their applications.

The Goals

- **Support direct communication between applications and scanners:** TWAIN Direct is not tied to any particular wire protocol, however, given the growing interest in mobile devices, the TWAIN Working Group is focusing TWAIN Direct on network connectivity. This network can be through a Cloud or as a direct communication between the user's device and the scanner. This design eliminates the need for vendor specific scanner drivers, reducing the support requirements for both scanner vendors and IT departments.

- **Simplify development:** Target the effort to the needs of the application writers. Adding basic image capture support to an application should take hours. Adding more complex support should only take days. The TWAIN Working Group supplies sample code in addition to the TWAIN Direct specification to help developers get started.

- **Support modern programming languages (not just C and C++):** Except for the image data, the command set in TWAIN Direct is composed of JSON data in the form of strings. The command set is human readable. The intention here is to make it easier for application writers who need to work with different programming languages for different mobile platforms.

- **Support the full functionality of scanners:** Seamlessly integrate custom features from one or more vendors within the same TWAIN Direct command set. Custom features are the driving force behind new additions to TWAIN Direct, so the standard makes it easy and safe to use them. The application can recommend certain scanners for the best user experience, while still being able to function with less capable scanners.

- **Provide applications access to all of the metadata associated with an image and seamlessly integrate custom metadata without risk of namespace collisions:**

The TWAIN Specification provides the starting point for standardized metadata items offered within TWAIN Direct.

- **Describe a baseline format for images that all scanners support, which can be easily expanded to encompass future functional requirements:** A proposal for PDF/raster is currently in-progress. The format will be verified as valid PDF, but constrained in such a way that application writers can extract the image data without having to use a PDF library. The format is also friendly to scanner vendors, who need to stream image data without altering the contents of the image itself.

- **Maintain version independence:** All versions of TWAIN Direct are interoperable across all applications and scanners. The TWAIN Working Group has more than 20 years of experience in this area with the TWAIN Specification.

- **Emphasize success:** Scanners must deliver images, unless the application states otherwise. For example, if an application *requests* JPEG compression, but the scanner cannot deliver it, then the scanner delivers the default (uncompressed raster data). If the application *requires* that JPEG compression is used, and that scanner cannot deliver it, then and only then will the scanner return a configuration error.

- **Anticipate the future:** Commonly used ether and Wi-Fi connections are not currently as fast as the best peripheral communication protocols (ex: USB 3.0). This will not always be the case. The expectation is that faster scanners will eventually migrate to this platform, and TWAIN Direct is designed to be ready.

To encourage adoption by application writers TWAIN Direct proposes a three-step path:

- Provide a way for legacy TWAIN scanners to benefit from TWAIN Direct, leveraging off the existing installed base.

- Demonstrate a way for existing USB scanners to expose a TWAIN Direct interface without modifying the base hardware (referred to as a sidecar solution). To accomplish this, the scanner is plugged into a small device that implements TWAIN Direct for it. This device could come from the scanner vendor or a third party provider.

- Encourage scanner vendors to natively support TWAIN Direct.

The Non-Goals

TWAIN Direct relies on industry standards and enablers in these areas. While these items are not defined by TWAIN Direct, recommendations for their use will be found within the associated sample code. Communication Managers which support these items may be found

in the appendices.

- Scanner discovery, registration and authentication.
- Communication security (when data is in motion).
- Image and metadata security (when data is at rest).

TWAIN Direct Syntax

JSON

TWAIN Direct is formatted using JSON. This specification does not go into detail about JSON, for that refer here: http://json.org. However, an experienced programmer should be able to effectively construct and parse the TWAIN Direct JSON formatted data without referencing any document other than this one.

Stylistic Conventions

TWAIN Direct is generally in agreement with the Google JSON Style Guide. The salient points are as follows:

- Property names are selected for the clarity of their meaning
- Property names are camelCased: the first word is lowercase, any subsequent words are capitalized, this applies to abbreviations: (ex: meterCpu, not meterCPU)
- The first character of a property name must be a lowercase alphabetic character in the range [a - z], subsequent characters may be alphabetic or numeric; no other characters are allowed
- All property names and values are encased within double quotes [U+0022 "]
- The property names for JSON arrays are plural (ex: actions)
- JavaScript keywords are reserved and may not be used for property names, this applies to both the TWAIN Direct standard and vendor customizations

Here is an example of the simplest practical TWAIN Direct Task:

```
{
  "actions": [
    {
      "action": "scan"
    }
  ]
}
```

TWAIN Direct Topology

Scanners provide images from one or more physical image capture elements built into their packaging. These capture elements may or may not be independently configurable. The collection of configurable elements within the scanner are referred to as the topology of the device.

This section examines the most common topologies. Understanding these concepts makes it easier to understand the rationale behind the format of a TWAIN Direct task.

Flatbed

This is the simplest scanner design, and the one seen most often on low end multifunction devices. Paper is placed on a glass surface. The user usually requests one kind of image format: black-and-white, grayscale or color. The scanner produces one image for each scan.

Some flatbed scanners support image segmentation for photographs or business cards. The user places one or more items on the glass surface. The scanner returns one image for each item. The takeaway from this for application developers is that a flatbed may produce more than one image.

Automatic Document Feeder (ADF)

With this design one or more sheets of paper are loaded into a feeder or a hopper. The user usually requests one kind of image format: black-and-white, grayscale or color.

Simplex feeders capture an image for just the front side of the sheet of paper. Duplex scanners return an image for the front and the rear of the sheet of paper. Some duplex scanners permit scanning from the rear only, and may support independent image processing settings, like color for the front of a sheet and grayscale for the rear.

ADF with Flatbed

An ADF and a flatbed are paired together, with all the features described above. In addition the scanner may automatically scan from the flatbed if paper is not loaded into the ADF's feeder or hopper.

Production Scanners

Production scanners utilize all the features described above. They may include automatic color detection and multiple pixelformats from a side of a sheet of paper.

With automatic color detection the scanner selects the best pixelformat for the image data that it has captured. In this case an application developer may opt to specify independent image processing settings for both color and grayscale, even though the scanner will only deliver one or the other.

With multi-format mode the scanner delivers more than one complete image for a side of a sheet of paper. A typical example is a color image for archival purposes and a black-and-white image for sending to an OCR engine to capture text.

Topology Breakdown

To cover the previous examples TWAIN Direct defines the following breakdown for a device topology:

action **AND** action **AND** action…
stream **OR** stream **OR** stream…
source **AND** source **AND** source…
pixelFormat **OR** pixelFormat **OR** pixelFormat...
attribute **AND** attribute **AND** attribute…
value **OR** value **OR** value...

An action specifies a command to a scanner. Actions are performed in order. All of the actions are attempted.

A stream selects a collection of sources. The first stream in an action that the scanner can support is selected, all others are ignored.

A source is a physical capture element, like the front and rear of a feeder, or a flatbed. All of the sources in a stream contribute to the flow of images returned from the scanner. A given source can appear more than once within the same stream.

A pixelFormat is a colorspace and a bit depth, such as rgb24. If more than one pixelFormat is specified within a source, then the scanner attempts to provide the best match for the captured image. If the scanner is unable to choose, then it selects the pixelFormat that captures the most information (ex: favoring rgb24 over gray8, and gray8 over bw1).

An attribute is a configurable item, such as compression or autocrop. All of the attributes are

attempted.

A value is applied to an attribute, such as jpeg to compression. The first value in an attribute that the scanner can support is selected, all others are ignored.

Topology Combinations

An application creates a topology with the goal of capturing image data in a certain way. The application decides on the complexity of the topology, and what constitutes failure, that is, under what conditions should the scanner report that it cannot perform the task. The scanner examines the topology of the task and determines what it's able to support. The default behavior is to ignore problems in the task and return an image.

With this in mind the application writer can do the following:

- Specify more than one stream. The most common use is to support capturing data from a flatbed if no feeder is available, or if the feeder does not have paper in it. It's may also be used to switch to the next stream if an exception requests it on a failed setting. Note that only one stream will be selected for a given action.

- Specify more than one source. For devices that support it, this is the way to select independent settings for feeder front, feeder rear and flatbed sources. It's also the way to create multi-pixelFormat output, such as capturing both color and grayscale images on one side of a sheet of paper.

- Specify more than one pixelFormat. In this case the scanner selects the best match for the image that it's captured. If the image contains enough color content, the scanner returns a color image, otherwise it will return grayscale or black-and-white.

- Most applications specify multiple attributes. As a general rule an application should only set attributes that it cares about, and rely on the default values for the rest.

- Specify more than one value for an attribute. The scanner selects the first supported value. This allows the application writer to test for specific values within an attribute without having to switch to a new stream.

TWAIN Direct Task

A TWAIN Direct task consists of one or more actions. Each action may have zero or more arguments. A TWAIN Direct task without any actions is a null task. A device reports success when it receives a null task, but takes no other action.

When a scanner accepts a task it implicitly locks the device for that application until the task is complete. Applications only run one task at a time. Ecosystems that come with session locks allow an application to run multiple tasks. A single task with multiple actions is functionally equivalent to several tasks run in order until the protection of a session lock.

The scanner runs every action specified in a task, unless the exceptions prevent it.

JSON

This figure shows the basic structure of a TWAIN Direct task in JSON format.

```
{
  "actions": [
    {
      "action": "#the name of this action#",
      "streams": [
        {
          "sources": [
            {
              "source": "#the name of an image source#",
              "pixelFormats": [
                {
                  "pixelFormat": "#the name of a pixel format#",
                  "attributes": [
                    {
                      "attribute": "#the name of an attribute#",
                      "values": [
                        { "value": "#a value#" }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

Pseudo-code

The JSON format of a TWAIN Direct task maps to the following data structure. Refer to the appendix for examples of TWAIN Direct tasks that can be used to fill this structure.

```
// The TWAIN Direct task
mandatory structure task

   // The actions for the task, evaluated in order of appearance
   mandatory array of structure actions

      // The identification of the action (ex: configure or scan)
      mandatory string action

      // The exception for this action
      optional string exception = { default is “nextObject” }

      // The vendor for this action
      optional string vendor = { default is TWAIN Direct UUID }

      // The streams for this action, evaluated in order of appearance
      mandatory array of structure streams

         // The exception for this stream
         optional string exception= { default is inherited from action }

         // The vendor for this stream
         optional string vendor = { default is inherited from action }

         // The sources for this stream, evaluated in order of appearance
         optional array of structure sources

            // Source of images (ex: any, feeder, feederFront, flatBed, etc)
            optional string source = { default is “any” }

            // The exception for this source
            optional string exception = { default is inherited from stream }

            // The vendor for this source
            optional string vendor = { default is inherited from stream }

            // The pixelformats for this source, evaluated in order of appearance
            optional array of structures pixelformat

               // Format of the image (ex: bw1, gray8, rgb24, etc)
               optional string pixelformat = { default set by device’s  “usedefault” }

               // The exception for this pixelFormat
               optional string exception = { default is inherited from source }

               // The vendor for this pixelFormat
               optional string vendor = { inherited from source }


               // The attributes for this pixelformat, in any order
```

```
            optional array of structure attributes

               // The identification of the attribute
               mandatory string attribute

               // The exception for this attribute
               optional string exception = { default is inherited from format }

               // The vendor for this attribute
               optional string vendor = { default is inherited from format }

               // One or more values for this attribute, attempted in order of appearance
               optional array of structure values

                  // A single value
                  mandatory string value

               end values
            end attributes
         end pixelformats
      end sources
   end streams
end actions
end task
```

Power-on Defaults

Vendors determine the power-on default settings for their scanners, which includes the default topology, and all associated attributes. These defaults are applied before processing any stream defined in a task. This way each stream can be trusted to build on the same settings for a given scanner model, without being affected by the settings of a previous stream.

To show this in action, consider a scanner that has both a feeder and a flatbed and only supports a pixelFormat of gray8. At power-on the scanner defaults to the flatbed.

This next task configures the scanner to use its power-on defaults, so the gray8 image comes from the flatbed.

```
{
  "actions": [
    {
      "action": "configure"
    }
  ]
}
```

This next task attempts to capture a color image from the feeder. The scanner cannot support this, so it moves to the next stream. When the second stream is entered the scanner resets to its power-on defaults. Note that the second stream does not specify a source. The end result is a gray8 image that comes from the flatbed.

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "source": "feeder",
              "pixelFormats": [
                {
                  "pixelFormat": "rgb24"
                }
              ]
            }
          ]
        },
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "pixelFormat": "gray8"
                }
              ]
```

```
                }
            ]
        }
    ]
  }
]
}
```

TWAIN Direct Errors

There are three kinds of errors that a scanner can encounter when trying to process a task:

- JSON syntax errors
- TWAIN Direct topology errors
- Unsupported values

A JSON syntax error prevents the scanner from processing the complete task. At a minimum it reports back the line number and character offset of the error. Scanner vendors may opt to return more information about the error, if they have it available.

A TWAIN Direct topology error occurs when topology entries appear out of order. For example, the following task is missing the stream object, which must appear within the action object, and which contains the sources object. In this case the scanner reports that the topology is in error.

```
{
  "actions": [
    {
      "action": "configure",
      "sources": [
        {
          "source": "feeder",
          "pixelFormats": [
            {
              "pixelFormat": "rgb24"
            }
          ]
        }
      ]
    }
  ]
}
```

Unsupported values can occur anywhere within a task. The way they are handled is determined by the exception system, which is described further below.

Exceptions

TWAIN Direct reduces the amount of code that an application needs to control a scanner. One important way of doing this is through the use of exceptions. The exception system allows an application to present choices to the scanner, which are used when the scanner is usable to configure itself in exactly the way the application wants.

Vendor marked items that are not recognized by the scanner are always ignored For instance, if one of the streams in a task is tagged with a “vendor” UUID, and that UUID does not match the current scanner, then that stream and all of its contents are skipped over and ignored.

The exception system offers the following options for each part of the topology.

action: ignore, fail, or nextAction
stream: ignore, fail, nextAction, or nextStream
source: ignore, fail, nextAction, or nextStream
pixelFormat: ignore, fail, nextAction, or nextStream
attribute: ignore, fail, nextAction, or nextStream
value: ignore, fail, nextAction, or nextStream (implied “next value”)

The mapping of default exceptions is laid out as follows:

actions: exception for all actions: **ignore**
stream:
exception for all streams except the last: **nextStream**
source: exception for all sources: **nextStream**
pixelFormat: exception for all pixelFormats: **nextStream**
attribute: exception for all attributes: **nextStream**
value: exception for last value: **nextStream**
...
exception for the last stream (use this if it’s the only stream): **ignore**
source: exception for all sources: **ignore**
pixelFormat: exception for all pixelFormats: **ignore**
attribute: exception for all attributes: **ignore**
value: exception for last value: **ignore**

Exceptions are inherited and can be overridden. For example, if nextStream is selected in the source it becomes the default for all pixelFormats, attributes and values under it, unless one of them specifies their own exception.

Exception: ignore

This is the default exception for TWAIN Direct tasks. The scanner ignores any configuration problems it encounters.

- The explicit use of “ignore” by any stream in a task means that the scanner will use that stream if it reaches it, any streams that follow will be ignored.
- The use of “ignore” by any source means that source will be included in the stream, if the scanner can support multiple sources.
- The use of “ignore” by any pixelFormat means the scanner will consider is as a candidate for the stream, if the stream supports automatic pixelFormat detection.
- Attributes that are not recognized are ignored.
- Values for attributes that are not recognized or cannot be set are ignored and the attribute retains its power-on default value..

Exception: fail

If during the course of processing a task the scanner runs into an item that it cannot recognize or support, and that item either inherited the “fail” exception from an outer portion of the topology or has it explicitly requested within the same item, then processing of the task stops and a failure is returned.

Application writers must be sparing in the use of the “fail” exception, reserving it for situations where returning an image that doesn’t exactly match the needs of the user is worse than returning no image at all.

Exception: nextAction

If during the course of processing a task the scanner runs into an item that it cannot recognize or support, and that item either inherited the “nextAction” exception from an outer portion of the topology or has it explicitly requested within the same item, then processing of the current action stops and the scanner proceeds to the next action.

If “nextAction” is specified for the last action in a list of actions, then it is treated an “ignore” exception.

Exception: nextStream

When there are multiple streams under an action, “nextStream” becomes the default exception for all except the last stream (or the only stream), which defaults to “ignore”.

If “nextStream” is specified, then an inability to recognize an item or to set a value causes processing of the current stream to stop, and the scanner proceeds to the next stream..

Exception: “next value”

This only applies to the values of an attribute and is always implied. The default behavior is to try each value in sequence. If none of the values can be used, then the exception for that attribute is applied.

In this example the application requests the scanner to set itself to a resolution of 50. If that’s not available a value of 75 is requested. If neither value can be set then the attribute and therefore the entire task is set to fail.

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "source": "any",
              "pixelFormats": [
                {
                  "pixelFormat": "rgb24",
                  "attributes": [
                    {
                      "attribute": "resolution",
                      "exception": "fail",
                      "values": [
                        { "value": 50 },
                        { "value": 75 }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    },
    {
      "action": "scan"
    }
  ]
}
```

PDF/raster Images

TWAIN Direct scanners produce finished images. Put another way, the image output from a scanner, if written to a file with the appropriate extension, is expected to be viewable by a program that recognizes that particular image file format.

TWAIN Direct scanners output image data using the following format:

PDF/raster (Portable Document Format)

PDF/raster is designed as a wrapper for 24-bit color and 8-bit grayscale as JPEG or uncompressed raster data, and 1-bit packed black-and-white as Group4 or uncompressed raster data.

The format is a valid PDF, but the use of a library is optional. The format is simple and consistent enough to allow a developer to open it and extract the image data.

Complete information on PDF/raster is available here: http://www.pdfraster.org

Metadata

Image metadata comes in four broad categories:

- Data describing the image, including the width, the height, the image format and the byte size of the image.
- Data describing the relationship of the image to other content, including the source of the image (feeder front, flatbed, etc) and the ordinal number of the sheet that supplied the image.
- Data gleaned from the image includes barcode, MICR and optical character recognition.
- Data that accompanies an image, such as text strings printed on the document by the scanner (if printed after scanning takes place these string will not appear on the image).

The data is organized as JSON, with objects grouping related properties. A typical example is shown below:

```
{
  "metadataTwainDirect": {
    "status": {
      "success": true
    },
    "address": {
      "imageNumber": 1,
      "sheetNumber": 1,
      "source": "feederFront"
    },
    "image": {
      "compression": "none",
      "pixelFormat": "bw1",
      "pixelHeight": 1650,
      "pixelOffsetX": 0,
      "pixelOffsetY": 0,
      "pixelWidth": 1280,
      "resolution": 150,
      "size": 265160
    }
    "imageBlock": {
      "imageNumber": 1,
      "imagePart": 1,
      "moreParts": "false",
    }
```

```
    }
}
```

---

metadataTwainDirect

Description

An object. The outermost wrapper for a collect of metadata objects. A scanner uses this object to give an application information about the image, how it was captured, and may include data that was found in the image.

Presence

Mandatory. All scanners must provide this object. Mandatory members of the object are marked with an asterisk (*). Items that are mandatory when working with **TBD** are marked with a double-asterisk (**).

Members

address*
barcode (TBD)
image*
imageBlock**
micr (TBD)
printer (TBD)
status*
vendor

Examples

```
{
  "metadataTwainDirect": {
    ...
  }
}
```

metadataTwainDirect.address

Description

An object. The members may be used to reconstruct the layout of the scanned sheets of paper.

Presence

Mandatory for all scanners. Mandatory members of the object are marked with an asterisk (*).

Members

imageNumber*
imagePart*
moreParts*
sheetNumber*
source*

Examples

```
{
  "metadataTwainDirect": {
    "address": {
        {
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.address.imageNumber

Description

An integer number. The first image after scanning begins must be 1, and each subsequent images increments by 1. The value is contiguous, and in combination with sheetNumber makes it possible for an application to detect discarded images as opposed to missing images, after scanning has concluded, just by examining the metadata.

In this example three sheets of paper were scanned. The second sheet was completely discarded, because the application asked to drop blank images. The application can detect this because of the gap in the sheetNumber. It knows that it has all of the images because the imageNumber counts by 1's without any breaks.

imageNumber	sheetNumber	source
1	1	feederFront
2	1	feederRear
-	-	-
-	-	-
3	3	feederFront
4	3	feederRear

Presence

Mandatory.

Values

1 - n	Integer values.

Examples

```
{
  "metadataTwainDirect": {
    "address": {
        {
          "imageNumber": 1,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.address.sheetNumber

Description

An integer number. The first sheet of paper after scanning begins must be 1, and each subsequent sheet increments by 1.

In this example three sheets of paper were scanned. The second sheet was completely discarded, because the application asked to drop blank images. The next image that the application gets has a sheetNumber of 3.

imageNumber	sheetNumber	source
1	1	feederFront
2	1	feederRear
-	-	-
-	-	-
3	3	feederFront
4	3	feederRear

Presence

Mandatory.

Values

1 - n	Integer values.

Examples

```
{
  "metadataTwainDirect": {
    "address": {
        {
          "sheetNumber": 1,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.address.source

Description A string indicating the source of the image.

Presence Mandatory.

Values

feederFront	The part of an automatic document feeder that scans the front of each sheet of paper. Scanners that only read one side of a sheet of paper must report feederFront.
feederRear	The part of an automatic document feeder that scans the rear of each sheet of paper.
flatBed	A glass surface that the paper is set upon.
planetary	A mounted camera, typically used for scanning books.
storage	An repository of images.

Examples

```
{
  "metadataTwainDirect": {
    "address": {
        {
          "source": "feederFront",
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image

Description

An object. The members provide information about the complete image.

Presence

Mandatory for all scanners, but only when metadataTwainDirect.address.moreParts reports false. Mandatory members of the object are marked with an asterisk (*).

Members

compression*
offsetX*
offsetY*
pixelFormat*
pixelHeight*
pixelOffsetX*
pixelOffsetY*
pixelWidth*
resolution*
size*

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.compression

Description A string indicating the form of compression used on the image.

Presence Mandatory.

Values

group4	CCITT FAX Group 4 for packed bitonal data (pixelFormat “bw1”).
jpeg	Standard JPEG for 8-bit grayscale and 24-bit color (pixelFormat “gray8” and “rgb24”).
none	Uncompressed raster data, suitable for all pixelFormat values.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "compression": "none",
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.offsetX

Description An integer value. It specifies the number of microns offset from the left (going along the x-axis) where the leftmost edge of the image was found.

Presence Mandatory.

Values

0 - n	The offset in microns.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "pixelOffsetX": 0,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.offsetY

Description

An integer value. It specifies the number of microns offset from the top (going along the y-axis) where the topmost edge of the image was found.

Presence

Mandatory.

Values

0 - n	The offset in microns.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "pixelOffsetY": 0,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.pixelFormat

Description A string. The colorspace and the bit depth of a pixel.

Presence Mandatory.

Values

bw1	Black-and-white with a bit depth of 1, also called packed bitonal, since an 8-bit byte contains 8 of these pixelFormats.
gray8	Grayscale with a bit depth of 8, allowing for 256 shades of grey.
gray16	Grayscale with a bit depth of 16, allowing for 65536 shades of grey.
rgb24	Color with a bit depth of 24 (8-bits per channel), allowing for 16.7 million colors..
rgb48	Color with a bit depth of 48 (16-bits per channel), allowing for 281 trillion colors..

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "pixelFormat": "bw1",
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.pixelHeight

Description

An integer value. It specifies the number of pixels measuring the distance from the topmost part of the image to the bottommost part.

Presence

Mandatory.

Values

1 - n	The complete height of the image in pixels.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "pixelHeight": 1650,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.pixelOffsetX

Description

An integer value. It specifies the number of pixels offset from the left (going along the x-axis) where the leftmost edge of the image was found.

Presence

Mandatory.

Values

0 - n	The offset in pixels.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "pixelOffsetX": 0,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.pixelOffsetY

Description

An integer value. It specifies the number of pixels offset from the top (going along the y-axis) where the topmost edge of the image was found.

Presence

Mandatory.

Values

0 - n	The offset in pixels.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "pixelOffsetY": 0,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.pixelWidth

Description

An integer value. It specifies the number of pixels measuring the distance from the leftmost part of the image to the rightmost part.

Presence

Mandatory.

Values

1 - n	The complete width of the image in pixels.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "pixelWidth": 1280,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.resolution

Description The resolution of the image in dots-per-inch (dpi).

Values

1 - n	An integer value. Typical values include but may not be limited to: 75, 100, 150, 200, 240, 250, 300, 400, 500, 600, 1200, 2400, 4800, 9600 and 19200.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "resolution": 150,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.image.size

Description The size of the complete image in bytes.

Values

1 - n	The size in bytes.

Examples

```
{
  "metadataTwainDirect": {
    "image": {
        {
          "size": 265160,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.imageBlock

Description

An object. The members are used to transfer images from the scanner to the application.

Presence

Mandatory for all scanners running on **TBD**. Mandatory members of the object are marked with an asterisk (*).

Members

imageNumber*
imagePart*
moreParts*

Examples

```
{
  "metadataTwainDirect": {
    "imageBlock": {
        {
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.imageBlock.imageNumber

Description

An integer number for the requested image block. The first image after scanning begins is always 1.

The imageNumber does not have to represent the entire image. Applications must examine the imagePart and moreParts members to know if they have all of the data for the image they are transferring.

Here is an example of two images. imageNumber 1 is transferred as a single block, imageNumber 2 is transferred using 3 blocks:

imageNumber	imagePart	moreParts
1	1	false
2	1	true
2	2	true
2	3	false

Presence

Mandatory.

Values

1 - n	Integer values, starting at 1 and incrementing by 1 for every image block.

Examples

```
{
  "metadataTwainDirect": {
    "address": {
        {
          "imageNumber": 1,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.imageBlock.imagePart

Description

An integer number identifying the part of an image that this image block represents. The first part for an imageNumber always begins with 1.

The scanner has the option to split an image into one or more parts. When this happens this number increments.

See metadataTwainDirect.address.imageNumber for more information.

Presence

Mandatory.

Values

1 - n	Integer values. Starting at 1 and incrementing by 1 for every image part.

Examples

```
{
  "metadataTwainDirect": {
    "address": {
        {
          "imagePart": 1,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.imageBlock.moreParts

Description

A boolean value. If false, then this is the last part of the image. If true, then the application must transfer more parts to get the complete image.

See metadataTwainDirect.address.imageNumber for more information.

Presence

Mandatory.

Values

false	This is the last part for this image.
true	There are more parts to this image after this part.

Examples

```
{
  "metadataTwainDirect": {
    "address": {
        {
          "moreParts": false,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.status

Description An object. The members provide the status of the image.

Presence Mandatory for all scanners. Mandatory members of the object are marked with an asterisk (*).

Members status*

Examples

```
{
  "metadataTwainDirect": {
    "status": {
        {
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.status.detected

Description

A string that marks a condition detected while capturing an image.

Presence

If no conditions are detected, then this property does not appear in the metadata.

This property is mandatory if “success” returns false.

Some scanners may be configured to provide this property when “success” is true; for instance, reporting that a successfully captured image has a folded corner.

Values

coverOpen	The scanner cover is in an open position.
foldedCorner	The image came from a sheet that has a folded corner.
imageError	a catch-all condition for imaging errors, such as low light levels from a lamp or an uncorrectable skew in the angle of the image.
misfeed	a catch-all condition for feeder errors, such as an inability to draw paper into the scanner.
multiFeed	Two or more sheets went through the scanner at the same time.
paperJam	The sheet of paper experienced a paper jam as the image was being captured.
staple	A staple was detected on the sheet of paper (or any item that could potentially damage the scanner).

Examples

```
{
  "metadataTwainDirect": {
    "status": {
        {
          "success": false,
          "detected": "paperJam",
          ...
```

```
            }
         }
      }
   }
}
```

metadataTwainDirect.status.success

Description A boolean indicating the status of the image. If true, then the image was successfully captured and scanning continues. If false, then an error was detected and scanning will end with the current sheet of paper.

Presence Mandatory.

Values

false	An error has been detected, see metadataTwainDirect.status.detected for more information.
true	The image was successfully captured.

Examples

```
{
  "metadataTwainDirect": {
    "status": {
        {
          "success": true,
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.vendor

Description An object. The members are determined by a scanner vendor. The contents of the object must be valid JSON following the TWAIN Direct Stylistic Conventions described in this document.

Presence Optional. Mandatory members are marked with an asterisk (*).

Members vendor*

Examples

```
{
  "metadataTwainDirect": {
    "vendor": {
        {
          ...
        }
      }
    }
  }
}
```

metadataTwainDirect.vendor.vendor

Description

A string. The UUID of the vendor defining the “vendor” metadata object. This UUID should be the same value as the one used when sending a task to the scanner.

The UUID is intended to help the application interpret the custom data received from the scanner, since the definition and meaning of similarly named properties can vary among vendors.

For instance, a value of “width” in the vendor object could be in units of inches, pixels or microns. Without consulting the UUID the application cannot be certain.

Presence

Mandatory.

Examples

```
{
  "metadataTwainDirect": {
    "vendor": {
        {
          "vendor": "C1528F4F-B6A2-46CA-A7B0-2C40BE74A5AB",
          ...
        }
      }
    }
  }
}
```

TBD: These are the outstanding items from TWAIN that need to be reviewed for inclusion in TWAIN Direct...

```
{ "sword":{
    "metadata":{
        "address":[
            { "data":{
                "id":"bookname",
                "value":"text"
            }},
            { "data":{
                "id":"chapternumber",
                "value":"1 - n"
            }},
               { "data":{
                "id":"documentnumber",
                "value":"1 - n"
            }},
            { "data":{
                "id":"imagemerged",
                "value":"yes/no"
            }},
            { "data":{
                "id":"imagenumber",
                "value":"1 - n"
            }},
            { "data":{
                "id":"patchcode",
                "value":"patch1/patch2/..."
            }},
            { "data":{
                "id":"sheetnumber",
                "value":"1 - n"
            }}
        ],
        "barcodelist":[
            { "barcode":[
                { "data":{
                    "id":"confidence",
                    "value":"0 - 100"
                }},
                { "data":{
                    "id":"rotation",
                     "value":"0 - 360"
                }},
                 { "data":{
                    "id":"text",
                    "value":"text"
                }},
                { "data":{
                    "id":"textlength",
                    "value":"0 - n"
                }},
```

```
                { "data":{
                    "id":"type",
                    "value":"2of5/30f9/..."
                }},
                { "data":{
                    "id":"xoffset",
                    "value":"0 - n"
                }},
                { "data":{
                    "id":"yoffset",
                    "value":"0 - n"
                }}
            ]}
        ],
        "deskew":[
            { "data":{
                "id":"anglefinal",
                "value":"0 - 360"
            }},
            { "data":{
                "id":"angleoriginal",
                "value":"0 - 360"
            }},
            { "data":{
                "id":"confidence",
                "value":"0 - 10"
            }},
            { "data":{
                 "id":"status",
                "value":"disabled/fail/reportonly/success"
            }},
            { "data":{
                "id":"windowx1",
                "value":"0 - n"
            }},
            { "data":{
                "id":"windowx2",
                   "value":"0 - n"
            }},
            { "data":{
                "id":"windowx3",
                "value":"0 - n"
            }},
            { "data":{
                "id":"windowx4",
                "value":"0 - n"
            }},
            { "data":{
                "id":"windowy1",
                "value":"0 - n"
            }},
            { "data":{
                "id":"windowy2",
                "value":"0 - n"
            }},
            { "data":{
```

```
                "id":"windowy3",
                "value":"0 - n"
            }},
            { "data":{
                "id":"windowy4",
                "value":"0 - n"
            }}
        ],
        "form":[
            { "data":{
                "id":"confidence",
         "value":"0 - 100"
            }},
            { "data":{
                "id":"horzdocoffset",
                "value":""
            }},
            { "data":{
                "id":"templatematch",
                "value":""
            }},
            { "data":{
                "id":"templatepagematch",
                "value":""
            }},
            { "data":{
                "id":"vertdocoffset",
                "value":""
            }}
        ],
        "image":[
            { "data":{
                "id":"pixelflavor",
                "value":"whiteonblack/blackonwhite"
            }},
            { "data":{
                "id":"planar",
                "value":"yes/no"
            }},
        ],
        "micrlist":[
            { "micr":[
                { "data":{
                    "id":"length",
                    "value":"0 - n"
                }},
                { "data":{
                    "id":"magdata",
                    "value":"text"
                }},
                { "data":{
                    "id":"magtype",
                    "value":"micr/raw/invalid"
                }}
            ]}
        ],
```

```
        "printerlist":[
            { "printer":[
                { "data":{
                    "id":"text",
                    "value":"text"
                }}
            ]}
        ]
    }
}}
```

Task and Action Objects

task

Description The outermost TWAIN Direct object.

Presence Mandatory.

If the object is empty the scanner returns success, but takes no other action. The task is complete after the status is returned.

Members actions

Examples

```
{
   ...
}
```

task.actions[ ]

Description

An array of one or more action objects for this task.

Presence

Optional. If not present the scanner returns success, but takes no other action. The task is complete after the status is returned.

Mandatory members are marked with an asterisk (*), and must be included if the actions array is present.

Members

action*
exception
streams
vendor

Examples

```
{
  "actions": [
    {
      ...
    }
  ]
}
```

task.actions[ ].action

Description A string. Indicates the action to take.

Presence Mandatory.

Values

configure	Configure the scanner.
null	The scanner returns success, but takes no other action
scan	Start scanning. If the Communication Manager has its own command for this, the scanner ignores this action.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      ...
    }
  ]
}
```

task.actions[ ].exception

Description A string. The exception to take if a problem occurs in the body of the current action.

Presence Optional. If not present each action in the array defaults to “nextObject”.

Values

failKey	Reject the entire task on an unsupported key
failKeyValue	Reject the entire task on an unsupported key or value
failValue	Reject the entire task on an unsupported value
ignore	Ignore the exception (accept the current action).
nextAction	Discard the current action and proceed to the next action in the array. If there is no next action, then this is interpreted as “fail”.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "exception": "nextObject",
      ...
    }
  ]
}
```

task.actions[ ].vendor

Description

A string. The UUID of the vendor defining the action. If the scanner recognizes the UUID then it examines the action. If it does not recognize it, then it ignores the entire action.

Presence

Optional. If not present the scanner defaults to the TWAIN Direct UUID, which is supported by all scanners.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "vendor": "C1528F4F-B6A2-46CA-A7B0-2C40BE74A5AB",
      ...
    }
  ]
}
```

Stream Object

task.actions[ ].streams[ ]

Description An array of one or more stream objects for a “configure” action. If the action isn’t set to “configure” the object is ignored.

Presence Optional. If not present the scanner scans uses its default stream. This default is determined by the scanner vendor.

Members

exception
sources
vendor

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          ...
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].exception

Description A string. The exception to take if a problem occurs in the body of the current stream.

Presence Optional. If not present the value is inherited from the action object.

Values

fail	Reject the entire task.
ignore	Ignore the exception (accept the current action).
nextAction	Discard the current action and proceed to the next action in the array. If there is no next action, then this is interpreted as “fail”.
nextObject	Discard the current object and proceed to the next object in the array. If there is no next object, then this is interpreted as “ignore”.
nextStream	Discard the current stream and proceed to the next stream in the array. If there is no next steam, then this is interpreted as “fail”.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "exception": "nextStream",
          ...
        },
        {
          "exception": "ignore",
          ...
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].vendor

Description

A string. The UUID of the vendor defining the stream. If the scanner recognizes the UUID then it examines the stream. If it does not recognize it, then it ignores the entire stream.

Presence

Optional. If not present the value is inherited from the action object.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "vendor": "C1528F4F-B6A2-46CA-A7B0-2C40BE74A5AB",
          ...
        }
      ]
    }
  ]
}
```

Source Object

task.actions[ ].streams[ ].sources[ ]

Description An array of one or more source objects for this stream.

Presence Optional. If not present the scanner scans using its default settings.

Members

exception
pixelFormats
source
vendor

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              ...
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].exception

Description A string. The exception to take if a problem occurs in the body of the current source.

Presence Optional. If not present the value is inherited from the stream object.

Values

fail	Reject the entire task.
ignore	Ignore the exception (accept the current action).
nextAction	Discard the current action and proceed to the next action in the array. If there is no next action, then this is interpreted as “fail”.
nextObject	Discard the current object and proceed to the next object in the array. If there is no next object, then this is interpreted as “ignore”.
nextStream	Discard the current stream and proceed to the next stream in the array. If there is no next steam, then this is interpreted as “fail”.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "exception": "fail",
              ...
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].source

Description A string that designates a distinct *source* of images within a scanner, such as the flatbed or ADF. Standard values are listed below.

Presence Optional. If not present, defaults to “any”.

Values

any	Any source at the scanner’s discretion, this includes custom sources that are not defined by the TWAIN Direct specification.
feeder	An automatic document feeder. This can be used with scanners that scan just one side or both sides of a sheet of paper.
feederFront	The part of an automatic document feeder that scans the front of each sheet of paper. If a scanner does not support independent control of feederFront and feederRear settings, then it uses feederFront for setting the entire scanner and ignores feederRear.
feederRear	The part of an automatic document feeder that scans the rear of each sheet of paper. If a scanner does not support independent control of feederFront and feederRear settings, then it uses feederRear if and only if it’s the only source in the stream.
flatBed	A glass surface that the paper is set upon.
planetary	A mounted camera, typically used for scanning books.
storage	An existing repository of images.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
```

```
              "source": "feeder",
              ...
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].vendor

Description

A string. The UUID of the owner of the source. If the scanner recognizes the UUID then it examines the source. If it does not recognize it, then it ignores the entire source.

Presence

Optional. If not present the value is inherited from the stream object.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "vendor": "C1528F4F-B6A2-46CA-A7B0-2C40BE74A5AB",
              ...
            }
          ]
        }
      ]
    }
  ]
}
```

PixelFormat Object

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ]

Description An array of one or more pixelFormat objects for this source.

Presence Optional. If not present the scanner scans using its default settings.

Members

attributes
exception
pixelFormat
vendor

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  ...
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].exception

Description A string. The exception to take if a problem occurs in the body of the current pixelFormat.

Presence Optional. If not present the value is inherited from the source object.

Values

fail	Reject the entire task.
ignore	Ignore the exception (accept the current action).
nextAction	Discard the current action and proceed to the next action in the array. If there is no next action, then this is interpreted as “fail”.
nextObject	Discard the current object and proceed to the next object in the array. If there is no next object, then this is interpreted as “ignore”.
nextStream	Discard the current stream and proceed to the next stream in the array. If there is no next steam, then this is interpreted as “fail”.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "exception": "fail",
                  ...
                }
              ]
            }
          ]
        }
      ]
    }
  ]
```

```
}
```

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].pixelFormat

Description A string. The colorspace and the bit depth of a pixel.

Presence Optional. If not present the default is at the discretion of the scanner.

Values

bw1	Black-and-white with a bit depth of 1, also called packed bitonal, since an 8-bit byte contains 8 of these pixelFormats.
gray8	Grayscale with a bit depth of 8, allowing for 256 shades of grey.
gray16	Grayscale with a bit depth of 16, allowing for 65536 shades of grey.
rgb24	Color with a bit depth of 24 (8-bits per channel), allowing for 16.7 million colors..
rgb48	Color with a bit depth of 48 (16-bits per channel), allowing for 281 trillion colors..

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "pixelFormat": "rgb24",
                  ...
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].vendor

Description

A string. The UUID of the owner of the pixelFormat. If the scanner recognizes the UUID then it examines the pixelFormat. If it does not recognize it, then it ignores the entire pixelFormat.

Presence

Optional. If not present the value is inherited from the source object.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "vendor": "C1528F4F-B6A2-46CA-A7B0-2C40BE74A5AB",
                  ...
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

Attribute Object

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].attributes[ ]

Description An array of one or more attribute objects for this pixelFormat.

Presence Optional. If not present the scanner scans using its default settings.

Members

attribute
exception
values
vendor

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      ...
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].attributes[ ].attribute

Description A string. The colorspace and the bit depth of a pixel.

Presence Optional. If not present the default is at the discretion of the scanner.

Values

Refer to the section on TWAIN Direct Attributes for the complete list

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "compression",
                      ...
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].attributes[ ].exception

Description A string. The exception to take if a problem occurs in the body of the current attribute.

Presence Optional. If not present the value is inherited from the pixelFormat object.

Values

fail	Reject the entire task.
ignore	Ignore the exception (accept the current action).
nextAction	Discard the current action and proceed to the next action in the array. If there is no next action, then this is interpreted as “fail”.
nextStream	Discard the current stream and proceed to the next stream in the array. If there is no next steam, then this is interpreted as “fail”.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "exception": "fail",
                      ...
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].attributes[ ].vendor

Description

A string. The UUID of the owner of the attribute. If the scanner recognizes the UUID then it examines the attribute. If it does not recognize it, then it ignores the entire attribute.

Presence

Optional. If not present the value is inherited from the pixelFormat object.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "vendor": "C1528F4F-B6A2-46CA-A7B0-2C40BE74A5AB",
                      ...
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

Value Object

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].attributes[ ].values[ ]

Description An array of one or more value objects for this attribute. The scanner selects the first value that it supports and ignores all others.

Presence Optional. If not present the scanner scans using its default settings.

Members value

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "values": [
                        {
                          ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

task.actions[ ].streams[ ].sources[ ].pixelFormats[ ].attributes[ ].values[ ].value

Description A string. A proposed value for this attribute.

Presence Optional. If not present the scanner scans using its default settings.

Values

Refer to the section on TWAIN Direct Attributes for the complete list

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "values": [
                        {
                          "value": "some value"
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

TWAIN Direct Attributes

Overview

This section describes all of the attributes supported by TWAIN Direct.

compression

Description

The compression that the scanner applies to the image. Use compression to reduce the amount of data coming out of the scanner, both to improve performance and to get smaller image files on disk.

Values

autoVersion1	Automatically selects group4 or jpeg compression based on the pixelFormat.
group4	CCITT Group 4 Fax, appears in the PDF/raster as /Filter /CCITTFaxDecode (with /K -1). For pixelFormat bw1 only.
jpeg	JPEG Baseline, appears in the PDF/Raster as /Filter /DCT. For pixelFormat rgb24 and gray8 only.
none	Uncompressed rasters, with each raster line aligned on a 4-byte boundary. Appears in the PDF/raster as /Filter null.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "compression",
                      "values": [
                        {
                          "value": "jpeg",
                          ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
```

```
      }
   ]
}
```

cropping

Description

Select the method used by the scanner to locate the region of interest, that is, the portion of the sheet of paper that the user wants to capture.

Values

auto	Automatically detect the cropping area for the full sheet of paper. If scanning from a flatbed the image will include all of the items found on the glass surface.
autoMultiple	Automatically detect the cropping area for the full sheet of paper for an automatic document feeder. If scanning from a flatbed the scanner will look for multiple images (such as photographs or business cards) and will return one image for each item that it finds.
fixed	Crop the image using the height/width/offsetX/offsetY attributes.
long	Used for long documents. Divide the image into multiple pieces using the height/width/offsetX/offsetY attributes. Each portion appears as its own /XObject in the finished PDF/raster.
relative	Find the borders of the full sheet of paper, and then select a region within that using the height/width/offsetX/offsetY attributes.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "cropping",
                      "values": [
                        {
                          "value": "auto",
```

```
                            ...
                        }
                    ]
                }
            ]
        }
    ]
}
```

height

Description The height of the captured image in microns.

Values

"integer value"	An integer value in microns. The conversion of inches to microns is (inches * 25400), so 11 inches is 279400 microns.
minimum	Use the minimum height supported by the scanner. This can be used in concert with a value of "maximum" for the offsetY.
maximum	Use the maximum height supported by the scanner, adjusted for the selected offsetY value. This can be used in concert with a value of "minimum" for the offsetY.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "height",
                      "values": [
                        {
                          "value": 279400,
                          ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

numberOfSheets

Description

The number of sheets that the scanner will capture. If the scanner runs out of sheets before reaching this number scanning will conclude.

Values

"integer value"	An integer value in sheets. The minimum value is 1.
maximum	Capture the maximum number of sheets supported by the scanner.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "numberOfSheets",
                      "values": [
                        {
                          "value": "maximum",
                          ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

offsetX

Description The horizontal offset of the captured image in microns.

Values

"integer value"	An integer value in microns. The conversion of inches to microns is (inches * 25400).
minimum	Use the minimum horizontal offset supported by the scanner. This can be used with a value of "maximum" for width.
maximum	Use the maximum horizontal offset supported by the scanner, adjusted for the selected width value if offsetx and width are in conflict. This can be used with a value of "minimum" for width.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "offsetX",
                      "values": [
                        {
                          "value": 25400,
                          ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

offsetY

Description The vertical offset of the captured image in microns.

Values

"integer value"	An integer value in microns. The conversion of inches to microns is (inches * 25400).
minimum	Use the minimum vertical offset supported by the scanner. This can be used with a value of "maximum" for height.
maximum	Use the maximum vertical supported by the scanner, adjusted for the selected height value if offsety and height are in conflict. This can be used with a value of "minimum" for height.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "offsetY",
                      "values": [
                        {
                          "value": 25400,
                          ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

resolution

Description The resolution of the image in dots-per-inch (dpi).

Values

"integer value"	An integer value. Typical values are 75, 100, 150, 200, 240, 250, 300, 400, 500, 600, 1200, 2400, 4800, 9600 and 19200.
optical	Use the optical resolution of the scanner. This produces the best image (least artifacts from scaling) that the scanner can produce, but the images can be large if the optical resolution is high, such as 1200dpi, which will result in a 404MB uncompressed color image for an 8.5 x 11 inch sheet of paper.
preview	Use the minimum resolution of the scanner suitable for creating preview images.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "resolution",
                      "values": [
                        {
                          "value": 300,
                          ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

width

Description The width of the captured image in microns.

Values

"integer value"	An integer value in microns. The conversion of inches to microns is (inches * 25400), so 8.5 inches is 215900 microns.
minimum	Use the minimum width supported by the scanner. This can be used in concert with a value of "maximum" for the offsetX.
maximum	Use the maximum width supported by the scanner, adjusted for the selected offsetX value. This can be used in concert with a value of "minimum" for the offsetX.

Examples

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "pixelFormats": [
                {
                  "attributes": [
                    {
                      "attribute": "width",
                      "values": [
                        {
                          "value": 215900,
                          ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

Appendix A: Sample Tasks

Null Task

The simplest TWAIN Direct task. The scanner confirms with success, but takes no other action.

```
{
}
```

Simplest Scan Task

The scanner configures itself to its default settings (settings after powering up) and begins scanning.

```
{
  "actions": [
    {
      "action": "scan"
    }
  ]
}
```

Automatic Pixel Format Detection

This configuration specifies two pixel formats for the source. For scanners that support this feature the scanner decides which format is more appropriate based on the amount of color content detected in the image. A color photo used rgb24, a monochrome document uses gray8.

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "source": "any",
              "pixelFormats": [
                {
                  "pixelFormat": "rgb24"
                },
                {
```

```
                  "pixelFormat": "gray8"
                }
              ]
            }
          ]
        }
      ]
    },
    {
      "action": "scan"
    }
  ]
}
```

Multi-Stream Scan Task

This configuration specifies two sources for the stream. For scanners that support this feature the user receives two images for each side of a sheet of paper: one color and one black-and-white.

```
{
  "actions": [
    {
      "action": "configure",
      "streams": [
        {
          "sources": [
            {
              "source": "any",
              "pixelFormats": [
                {
                  "pixelFormat": "rgb24"
                }
              ]
            },
            {
              "source": "any",
              "pixelFormats": [
                {
                  "pixelFormat": "bw1"
                }
              ]
            }
          ]
        }
      ]
    },
    {
      "action": "scan"
    }
  ]
}
```