TWAIN Direct™ Specification: Metadata

Ratified October 2nd 2017
Revision 1.0

History

Date	Version	Comment
September 15th, 2017	1.00	First version

Notes

Notes
• (none)

Contents

History

Notes

Contents

Glossary of Terms

References

PDF/raster

Images and Numbering
- One Small Image
- One Large Image
- One Very Large Image
- Multiple Images from a Single Sheet
- Multiple Sheets
- Discarding Blank Images

Metadata
- metadata
- metadata.address
- metadata.address.imageNumber
- metadata.address.imagePart
- metadata.address.moreParts
- metadata.address.pixelFormatName
- metadata.address.sheetNumber
- metadata.address.source
- metadata.address.sourceName
- metadata.address.streamName
- metadata.barcodes
- metadata.barcodes[ ].base64Data
- metadata.barcodes[ ].pixelOffsetX
- metadata.barcodes[ ].pixelOffsetY
- metadata.barcodes[ ].type
- metadata.image
- metadata.image.compression

metadata.image.imageMerged
metadata.image.pixelFormat
metadata.image.pixelHeight
metadata.image.pixelOffsetX
metadata.image.pixelOffsetY
metadata.image.pixelWidth
metadata.image.resolution
metadata.micr
metadata.micr.base64Data
metadata.micr.type
metadata.patchCode
metadata.patchCode.type
metadata.status
metadata.status.detected
metadata.status.success
metadata.vendors
metadata.vendors[ ].vendor

Glossary of Terms

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

Word	Meaning
action	A TWAIN Direct command (e.g. “configure”).
application	A program that sends TWAIN Direct commands to a scanner.
attribute	A configurable item, such as compression, resolution, 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.
pixelFormat	The combination of a color space 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.
topology	The combination in a configure action of a stream, source and pixelFormat used to address components within the scanner.
user	A person in control of an application and a scanner.

References

This section lists standards, guides and resources cited in this document.

Word	Meaning
Base64	Refer to 5.2 Base64 Content-Transfer-Encoding http://www.w3.org/Protocols/rfc1341/5_Content-Transfer-Encoding.html
Google JSON Style Guide	Google JSON Style Guide https://google-styleguide.googlecode.com/svn/trunk/jsoncstyleguide.xml
JavaScript Reserved Words	List of reserved words http://www.w3schools.com/js/js_reserved.asp
JSON	RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON) https://www.ietf.org/rfc/rfc4627.txt TWAIN Direct requires all task content to be contained in an object token. With this restriction in place the following JSON parsers may also be used. Just confirm that the outmost token is an object before proceeding. ECMA-404 - http://www.json.org RFC 7159 - http://www.rfc-editor.org/rfc/rfc7159.txt A convenient tool to compact, beautify, and validate JSON data https://jsonformatter.curiousconcept.com/
PDF/raster	PDF Raster Documents http://pdfraster.org
TWAIN DIrect Sample Code	Repository for TWAIN Direct sample code https://github.com/twain/twain-direct
TWAIN Direct	Website for TWAIN Direct http://twaindirect.org
TWAIN Direct UUID Version 1	**211a1e90-11e1-11e5-9493-1697f925ec7b** https://www.uuidgenerator.net (generation source)
UUID	A Universally Unique IDentifier (UUID) URN Namespace http://www.ietf.org/rfc/rfc4122.txt
W3C RDF Validation Service	Resource Description Framework (RDF) - this website can be used to check and visualize RDF documents. https://www.w3.org/RDF/Validator/

PDF/raster

A PDF/raster image generated by a TWAIN Direct scanner must include the metadata for the imageBlock that has a moreParts of “lastPartInFile” for a complete image, or “lastPartInFileMorePartsPending” for a part of an image. The metadata is included at the page level of the PDF/raster file, and uses the following wrapper:

```
<?xpacket begin="?" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpdata xmlns:x="adobe:ns:meta/">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:twaindirect="http://www.twaindirect.org/twaindirect">
<rdf:Description rdf:about="http://www.twaindirect.org/twaindirect#metadata">
<twaindirect:metadata>
BASE64(TwainDirectMetadata)
</twaindirect:metadata>
</rdf:Description>
</rdf:RDF>
[optional additional <rdf:RDF></rdf:RDF> content]
</x:xmpdata>
<?xpacket end="w"?>
```

The scanner embeds its JSON formatted metadata as a BASE64 string within the <twaindirect:metadata> tag.

Vendors may add additional RDF content in addition to that supplied by TWAIN Direct, as indicated in the sample shown above. There is a link to a validation service in the References section of this document.

Images and Numbering

TWAIN Local uses an image block number to index an imageBlock. An imageBlock represents a portion of PDF/raster data transferred from a scanner to an application. A single PDF/raster file may be transferred using one or more image blocks.

In most cases a single PDF/raster file represents a complete image. In some cases (very long documents or very high resolutions) multiple PDF/raster files may be needed to represent a complete image.

One of the goals of TWAIN Direct is to supply an application with sufficient metadata that it can reconstruct the original batch that the user scanned. Every image can be associated with every sheet of paper, and every sheet of paper can be accounted for.

sheetNumber counts the total physical sheets of paper. It starts at 1 (set when the scanner receives the startCapturing command) and increments for every sheet of paper.

imageNumber counts the total images. It starts at 1 (set when the scanner receives the startCapturing command). It is incremented when moreParts is set to lastPartInFile.

imagePart counts the number of PDF/rasters files for a given imageNumber. It starts at 1 (set when the scanner receives the startCapturing command). It is incremented when moreParts is set to lastPartInFileMorePartsPending. It is reset back to 1 when moreParts is set to lastPartInFile.

Several examples are provided to illustrate how these numbers work.

One Small Image

In this example one sheet of paper is scanned from a flatbed at a low enough resolution for it to completely fit inside one imageBlock.

image Block	image Number	sheet Number	image Part	more Parts	source
1	1	1	1	lastPartInFile	flatbed

One Large Image

In this example one sheet of paper is scanned from a flatbed at a high enough resolution that it requires three imageBlocks to fully transfer the image.

The output file is (numbers are sheet, image, and part):

file-01-01-01-flatbed.pdf

image Block	image Number	sheet Number	image Part	more Parts	source
1	1	1	1	morePartsPending	flatbed
2	1	1	1	morePartsPending	flatbed
3	1	1	1	lastPartInFile	flatbed

One Very Large Image

In this example one long sheet of paper is scanned from a feeder at a very high resolution that requires many imageBlocks and three complete PDF/rasters to fully transfer the single image.

The output files are (numbers are sheet, image, and part):

file-01-01-01-flatbed.pdf
file-01-01-02-flatbed.pdf
file-01-01-03-flatbed.pdf

image Block	image Number	sheet Number	image Part	more Parts	source
1	1	1	1	morePartsPending	feederFront
2	1	1	1	morePartsPending	feederFront
3	1	1	1	lastPartInFileMorePartsPending	feederFront
4	1	1	2	morePartsPending	feederFront
5	1	1	2	morePartsPending	feederFront
6	1	1	2	lastPartInFileMorePartsPending	feederFront
7	1	1	3	lastPartInFile	feederFront

Multiple Images from a Single Sheet

In this example multiple images are returned from a single sheet of paper scanned duplex (front and rear). Some images are color and some are black-and-white, so some require more imageBlocks to transfer than others.

The output files are (numbers are sheet, image, and part):

file-01-01-01-front.pdf
file-01-02-01-front.pdf
file-01-03-01-rear.pdf
file-01-04-01-rear.pdf

image Block	image Number	sheet Number	image Part	more Parts	source
1	1	1	1	lastPartInFile	feederFront
2	2	1	1	morePartsPending	feederFront
3	2	1	1	morePartsPending	feederFront
4	2	1	1	lastPartInFile	feederFront
5	3	1	1	lastPartInFile	feederRear
6	4	1	1	morePartsPending	feederRear
7	4	1	1	morePartsPending	feederRear
8	4	1	1	lastPartInFile	feederRear

Multiple Sheets

In this example three sheets of paper are scanned duplex (so front and rear), some sheets require more imageBlocks than others, which can happen in a mixed batch of long and short documents.

The output files are (numbers are sheet, image, and part):

file-01-01-01-front.pdf
file-01-02-01-rear.pdf
file-02-03-01-front.pdf
file-02-04-01-rear.pdf
file-03-05-01-front.pdf
file-03-06-01-rear.pdf

image Block	image Number	sheet Number	image Part	more Parts	source
1	1	1	1	lastPartInFile	feederFront
2	2	1	1	lastPartInFile	feederRear
3	3	2	1	morePartsPending	feederFront
4	3	2	1	lastPartInFile	feederFront
5	4	2	1	morePartsPending	feederRear
6	4	2	1	lastPartInFile	feederRear
7	5	3	1	lastPartInFile	feederFront
8	6	3	1	lastPartInFile	feederRear

Discarding Blank Images

In this example four sheets of paper are scanned duplex (so front and rear). The scanner has been asked to discard blank images. The second sheet of paper is blank on its front. The third sheet of paper is blank on both front and rear.

The output files are (numbers are sheet, image, and part):

file-01-01-01-front.pdf
file-01-02-01-rear.pdf
file-02-03-01-rear.pdf
file-04-04-01-front.pdf
file-04-05-01-rear.pdf

image Block	image Number	sheet Number	image Part	more Parts	source
1	1	1	1	lastPartInFile	feederFront
2	2	1	1	lastPartInFile	feederRear
3	3	2	1	lastPartInFile	feederRear
4	4	4	1	lastPartInFile	feederFront
5	5	4	1	lastPartInFile	feederRear

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 (feederFront, flatbed, etc) and the ordinal number of the sheet that supplied the image. This information can be used to reconstruct the organization of the sheets of paper captured by the scanner.

- Data gleaned from the image, including barcode, MICR and patch codes.

- 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). Printing is TBD as of this writing (14-Aug-2016).

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

```
{
  "metadata": {
    "status": {
      "success": true
    },
    "address": {
      "imageNumber": 1,
      "imagePart": 1,
      "moreParts": "lastPartInFile",
      "sheetNumber": 1,
      "source": "feederFront",
      "streamName": "stream0",
      "sourceName": "source0",
      "pixelFormatName": "pixelFormat0"
    },
    "image": {
      "compression": "none",
      "pixelFormat": "bw1",
      "pixelHeight": 1650,
      "pixelOffsetX": 0,
      "pixelOffsetY": 0,
      "pixelWidth": 1280,
      "resolution": 150
    }
  }
```

```
}
```

metadata

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 properties of the object are marked with a one ([1]).

Members

address[1]
barcodes
image[1]
micr
patchCode
status[1]
vendors

Examples

```
{
  "metadata": {
    ...
  }
}
```

metadata.address

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

Presence Mandatory for all scanners. All properties are mandatory.

Members

imageNumber
imagePart
moreParts
pixelFormatName
sheetNumber
source
sourceName
streamName

Examples

```
{
  "metadata": {
    "address": {
      ...
    }
  }
}
```

metadata.address.imageNumber

Description

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

Refer to the Images and Numbering section for examples.

Presence

Mandatory.

Values

1 - n	Integer values. Starting at 1 and incrementing by 1 for every complete image. Resets to 1 when the startCapturing command is accepted.

Examples

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

metadata.address.imagePart

Description

An integer number identifying the part of an image that this image block represents. The first part for an imageNumber always starts at 1. All of the imageBlocks for for a given imagePart number represent a single PDF/raster file.

Refer to the Images and Numbering section for examples.

Presence

Mandatory.

Values

1 - n

Integer values. Starting at 1 and incrementing by 1 for every image part. The value is set to 1 when the startCapturing command is accepted. It is incremented by 1 for the next image part when moreParts is set to lastPartInFileMorePartsPending. It is reset back to 1 for the next image part when moreParts is set to lastPartInFile.

Examples

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

metadata.address.moreParts

Description A string indicating if this imagePart is the last part in this file, or if more parts are needed to complete the image.

Refer to the Images and Numbering section for examples.

Presence Mandatory.

Values

lastPartInFile This is the last part for this image.

lastPartInFileMorePartsPending
This is the last part in this file, but not in this image.

morePartsPending There are more parts to this image after this part.

Examples

```
{
  "metadata": {
    "address": {
      "moreParts": "lastPartInFile",
      ...
    }
  }
}
```

metadata.address.pixelFormatName

Description A string. The name of the pixelFormat in the task associated with this image. Empty if a pixelFormat was not specified.

Presence Mandatory.

Values Any valid UTF-8 encoded JSON string.

Examples

```
{
  "metadata": {
    "address": {
      "pixelFormatName": "pixelFormat0",
      ...
    }
  }
}
```

metadata.address.sheetNumber

Description

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

Refer to the Images and Numbering section for examples.

Presence

Mandatory.

Values

1 - n	Integer values. Starting at 1 when the startCapturing command is accepted, and incrementing by 1 for every sheet of paper captured by the scanner.

Examples

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

metadata.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

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

metadata.address.sourceName

Description A string. The name of the source in the task associated with this image. Empty if a source was not specified.

Presence Mandatory.

Values Any valid UTF-8 encoded JSON string.

Examples

```
{
  "metadata": {
    "address": {
      "sourceName": "source0",
      ...
    }
  }
}
```

metadata.address.streamName

Description A string. The name of the stream in the task associated with this image. Empty if a stream was not specified.

Presence Mandatory.

Values Any valid UTF-8 encoded JSON string.

Examples

```
{
  "metadata": {
    "address": {
      "streamName": "stream0",
      ...
    }
  }
}
```

metadata.barcodes

Description An array. Each object in the array describes a single barcode found on the image.

Presence Mandatory for scanners that support the barcodes attribute. All properties are mandatory.

Members

base64Data
pixelOffsetX
pixelOffsetY
type

Examples

```
{
  "metadata": {
    "barcodes": [
        {
          ...
        }
      }
    ]
  }
}
```

metadata.barcodes[ ].base64Data

Description The barcode data in Base64 format. While many barcode formats return simple ASCII text, there are many that can return binary data, so it's easier for an application if it always has to decode the data that it receives.

Presence Mandatory.

Values String.

string Data encoded in the Base64 format.

Examples

```
{
  "metadata": {
    "barcodes": [
      {
        "base64Data": "U2FtcGxlIGRhdGEuLi4=",
        ...
      }
    ]
  }
}
```

metadata.barcodes[ ].pixelOffsetX

Description The X-offset of the barcode.

Presence Mandatory.

Values Number (positive integer).

0 - n The X-offset of the upper-left corner of the barcode, measured in pixels.

Examples

```
{
  "metadata": {
    "barcodes": [
      {
        "pixelOffsetX": 100,
        ...
      }
    ]
  }
}
```

metadata.barcodes[ ].pixelOffsetY

Description The Y-offset of the barcode.

Presence Mandatory.

Values Number (positive integer).

0 - n The Y-offset of the upper-left corner of the barcode, measured in pixels.

Examples

```
{
  "metadata": {
    "barcodes": [
      {
        "pixelOffsetY": 100,
        ...
      }
    ]
  }
}
```

metadata.barcodes[ ].type

Description The barcode type.

Presence Mandatory.

Values String. Refer to the barcode attribute for the full list of barcodes defined by TWAIN Direct.

Examples

```
{
  "metadata": {
    "barcodes": [
      {
        "type": "3Of9",
        ...
      }
    ]
  }
}
```

metadata.image

Description

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

Presence

Mandatory for all scanners, but only when metadata.address.moreParts reports lastPartInFile, indicating that the entire image has been captured and transferred. Mandatory members of the object are marked with a one ([1]).

Members

compression[1]
imageMerged
pixelFormat[1]
pixelHeight[1]
pixelOffsetX[1]
pixelOffsetY[1]
pixelWidth[1]
resolution[1]
size[1]

Examples

```
{
  "metadata": {
    "image": {
      ...
    }
  }
}
```

metadata.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

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

metadata.image.imageMerged

Description Indicates that the current image is the result of a merger between the front and rear images of single sheet of paper.

Presence Mandatory, if imageMerge was specified by the task and accepted by the scanner.

Values String, one of the following.

merged	The front and rear images were merged.
notMerged	This image has not been merged. This is the default, if imageMerged is not specified in the metadata.

Examples

```
{
  "metadata": {
    "image": {
      "imageMerged": "notMerged",
      ...
    }
  }
}
```

metadata.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

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

metadata.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. If the size of the value exceeds 2147483647, then the value must be sent as a string.

Presence Mandatory.

Values

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

Examples

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

metadata.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. If the size of the value exceeds 2147483647, then the value must be sent as a string.

Presence

Mandatory.

Values

0 - n	The offset in pixels.

Examples

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

metadata.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. If the size of the value exceeds 2147483647, then the value must be sent as a string.

Presence

Mandatory.

Values

0 - n	The offset in pixels.

Examples

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

metadata.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. If the size of the value exceeds 2147483647, then the value must be sent as a string.

Presence

Mandatory.

Values

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

Examples

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

metadata.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

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

metadata.micr

Description An object describing micr data found on the image.

Presence Mandatory for scanners that support the micr attribute. Mandatory members of the object are marked with a one ([1]).

Members base64Data[1]
type[1]

Examples

```
{
  "metadata": {
    "micr": {
      ...
    }
  }
}
```

metadata.micr.base64Data

Description The micr data in Base64 format.

Presence Mandatory.

Values String.

string Data encoded in the Base64 format.

Examples

```
{
  "metadata": {
    "micr": {
      "base64Data": "U2FtcGxlIGRhdGEuLi4=",
      ...
    }
  }
}
```

metadata.micr.type

Description The micr type.

Presence Mandatory.

Values String, one of the following.

invalid	Invalid MICR data.
micr	Valid MICR data.
raw	Raw unprocessed data.

Examples

```
{
  "metadata": {
    "micr": {
      "type": "micr",
      ...
    }
  }
}
```

metadata.patchCode

Description An object describing the patch code found on the image.

Presence Mandatory for scanners that support the patchCode attribute. Mandatory members of the object are marked with a one ([1]).

Members type[1]

Examples

```
{
  "metadata": {
    "patchCode": {
      ...
    }
  }
}
```

metadata.patchCode.type

Description The patch code type.

Presence Mandatory.

Values String. Refer to the patchCode attribute for the full list of patch codes defined by TWAIN Direct.

Note that patchT is never returned, because it's interpreted as a patch2 or a patch3. And patch4 is generally not returned, since it's used to change behavior in the scanner.

Examples

```
{
  "metadata": {
    "patchCode": {
      "type": "patch2"
    }
  }
}
```

metadata.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 a one ([1]). Mandatory members of the object that must be present if success is false are marked with a two ([2]).

Members

detected[2]
success[1]

Examples

```
{
  "metadata": {
    "status": {
      ...
    }
  }
}
```

metadata.status.detected

Description

A string that marks a condition detected while capturing the image associated with this imageBlock.

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. Applications should always test for the presence of detected.

Some image capture errors may not create an imageBlock, so there will be no metadata. These errors are reported in the RESTful API session object. Refer to *results.session.status* in the TWAIN Local and TWAIN Cloud documentation for more information.

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

```
{
  "metadata": {
    "status": {
      "success": false,
      "detected": "paperJam"
    }
  }
}
```

metadata.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 metadata.status.detected for more information.
true	The image was successfully captured.

Examples

```
{
  "metadata": {
    "status": {
      "success": true
    }
  }
}
```

metadata.vendors

Description

An array of objects. 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.

The use of an array allows vendors to chain additional data without affecting prior content. As a general rule if a given property appears more than once in a chain of vendor objects, the first one is the one the application must use. This allows vendors processing the data to supercede metadata that proceeded them, without being forced to modify the data provided by prior vendors. It also provides a history of changes that can be useful for diagnostics.

Presence

Optional. Mandatory members are marked with a one ([1]).

Members

vendor[1]

Examples

```
{
  "metadata": {
    "vendors": [
      {
        ...
      }
    ]
  }
}
```

metadata.vendors[ ].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

```
{
  "metadata": {
    "vendors": [
      {
        "vendor": "C1528F4F-B6A2-46CA-A7B0-2C40BE74A5AB",
        ...
      }
    ]
  }
}
```