X9Ware LLC

White Paper
On
EOFB Validation

Revision Date: 08/25/2022
Release R4.10

X9Ware LLC
10753 Indian Head Industrial Blvd
St Louis, Missouri 63132-1101
(844) 937-1850

Email: sales@x9ware.com

EOFB

Overview

EOFB (end of facsimile block) testing has created it's fair share of confusion and consternation within the Check21 image exchange community. As part of this information, we hope to provide some insight into proper EOFB recognition and validation.

Before we go deeper into this topic, we should first level set on exactly what EOFB is. From Check21 day one, the X9 image exchange standards require that all check images be encoded in TIFF format and that all image compression be done using the Group 4 Fax compression algorithm.

EOFB is a defined requirement as part of the x9.100-181 image standard which applies directly to the current x9.100-187+UCD specification, and presumably to the Canadian Standard 015 as well. EOFB does not otherwise currently apply directly to x9 image exchange.

EOFB is part of the Group 4 compression standard, and hence is mandated as part of Check21 image exchange. There is really no wiggle room in that definition. However, since TIFF images can be successfully processed without EOFB, almost all Check21 processors have historically accepted image exchange files without the presence of EOFB within the images. Because of that, EOFB has typically been more of a follow up conversion regarding image files, and not associated with hard failures and rejected files.

However, that situation is beginning to change. There has been more and more awareness of this topic within the industry. Proper inclusion of EOFB is certainly seen now as a best practice. It is becoming clear that a day may come in the future when EOFB is mandated by the major image exchange processors. It will only take several of the largest processors to go this way. If that happens, there could be a tide that flows through the industry where EOFB becomes a mandate.

Why would EOFB become an absolute mandate? There are several reasons. The first and most obvious is that it is a Group 4 Fax compression requirement, hence it is really required to be present in all TIFF images that are exchanged using the Check21 network. This position is strengthened by the firm statements within the x9.100-181 standard that EOFB is required. The second reason is that EOFB is a good indication that an image has been correctly formed and that it is entirely present. EOFB was devised as part of the Group 4 Fax standard since those images are typically transmitted by fax machines, and some indication was needed that the complete image had been both transmitted and received. The binary text within the image does not explicitly accomplish that, hence EOFB was incorporated into that standard.

What is EOFB?

When you get down to basics, each TIFF image contain one or more image segments. Each image segment is individually defined by an offset within the image (tag 273) and a data length (tag 279). Each segment contains a compressed version of 1's and 0's that represent the black and white pixels within the image itself. As an aside, tiff tag 262 (photometric interpretation) is defined as zero when the 0's represent white and as one when the 0's represent black.

Per the Group 4 Fax definition, each image segment must include an "end-of-facsimile-block" indicator at the logical end of the segment data. This is a marker that the image segment is complete and fully accounted for. Without the EOFB marker, you do not have absolute knowledge that the image segment is complete. TIFF readers many times may assume that missing rows consist entirely of white pixels. EOFB is a positive marker that image segment is present in it's entirety.

X9 Standards

Since EOFB issues can be challenging and take considerable time and effort to resolve, there is a strong argument that this validation should always be performed so conformance can be confirmed.

Per the X9.100-187 standard:

- Each G4 compressed image strip shall be terminated by a 24-bit End-of-Facsimile-Block (EOFB) code. The EOFB code is defined in the iTU-T Recommendation T.6 Specification.
- TIFF writers shall always encode the full, prescribed number of rows, with a proper EOFB immediately following in the encoding. Padding shall be by the least number of 0-bits needed for the T.6 encoding to exactly occupy a multiple of 8 bits. Only 0-bits shall be used for padding, and StripByteCount shall not extend to any bytes not containing properly formed T.6 encoding.

Note that the standard does require that the required pad bits be zero. Some encoders will correctly append the EOFB but then do not ensure that the subsequent pad bits are zero. This is incorrect per the x9.100-187 standard definition.

What does EOFB look like?

EOFB has the following definition and characteristics:

- EOFB is the three byte (24 bit) sequence x'001001'.
- Each image segment must have the EOFB sequence appended at the end.
- Each image segment is then padded with binary zeroes to fill out the last eight bit byte.
- Construction thus requires that from zero to seven pad bits be appended after the EOFB to fill out the last physical byte.
- When done correctly, the last byte within the segment data cannot be zero.

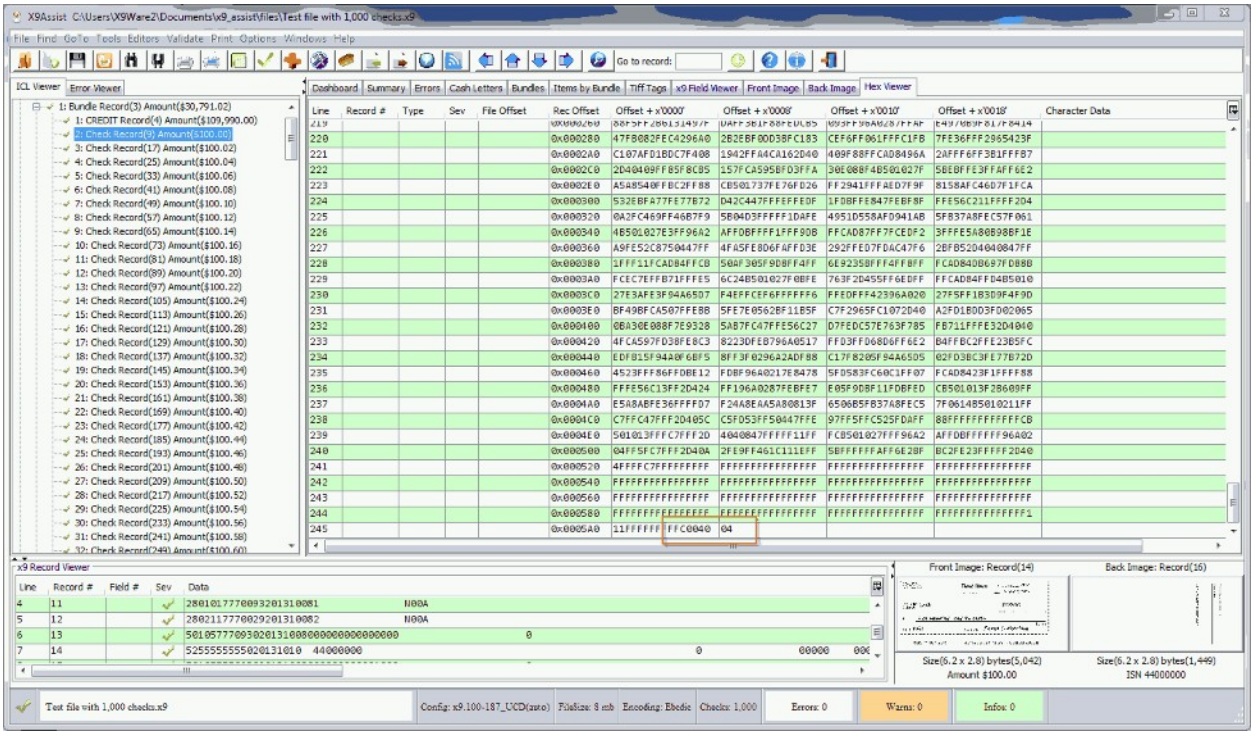
As previously described, EOFB should end with a 24 bit sequence of x'001001' that is then padded with up to 7 zero bits to fill out the last byte within the image segment. This means that the tiff image segment must end with one of the following (in hex) to be valid per Group 4 Fax specifications:

x'001001'	no bit padding required
x'002002'	logical pad of 1 bit at the end of the image segment
x'004004'	logical pad of 2 bits at the end of the image segment
x'008008'	logical pad of 3 bits at the end of the image segment
x'0010010'	logical pad of 4 bits at the end of the image segment
x'0020020'	logical pad of 5 bits at the end of the image segment
x'0040040'	logical pad of 6 bits at the end of the image segment
x'0080080'	logical pad of 7 bits at the end of the image segment

The following is an example of what a TIFF image segment can look like in a hex viewer with a correctly formed EOFB. In the case provided, the EOFB marker has a value of x'004004" which is the third in the above provided list.

All of this starts to make things appear simple. Now for the complications:

- As stated earlier, there can be more than one image segment that is present within a tiff image. EOFB validation requires that every image segment be checked.
- The TIFF IFD (Image File Directory) typically appears at the front of an image with the image segment(s) appearing at the end of the image. However, that is not a requirement and many image encoders will put the IFD at the end of the image and not at the beginning. The atsk of visually inspecting the EOFB is much more complex when the IFD is located at the end of the image (and not at the beginning).



Validation Process

Validating the correct presence of EOFB requires inspection of the last **three or four bytes** of each image segment. Essentially, this inspection can be performed as follows:

- Isolate the last four bytes of image segment data.
- Continue shifting the isolated image data right, one bit at a time, until the bottom most right bit is non-zero. This process takes advantage of the requirement that the pad bits always be zero and that the 24 bit EOFB sequence ends in a non-zero bit.
- Note that no more than seven (7) shifts should be necessary.
- Validate that the resulting right most bit sequence in the rotated data is x'001001'.

How can your x9 tool provide good EOFB support?

The follows tools and capabilities really help:

1. Ability to enable or disable EOFB validation at the x9 specification level
2. Ability to flag EOFB errors at varying severity levels as you implement full validation
3. Ability to see images in hex format
4. Ability to see a parsed version of TIFF directory
5. Ability to export EOFB validation errors so they can be easily reported to the originator
6. Ability to export failed images so they can be provided to your originators
7. Ability to activate debug logging to provide more insight into failed EOF validations
8. Ability to export tracing information so you can forward to your originators

EOFB Logging

EOFB testing can be difficult. To assist in your testing and validation, it is important that your image validation tool provide feedback which allows you to easily determine image content so you can see why the image has failed EOFB testing. Without this, you believe that the image has failed EOFB but you have no proof of that and nothing to confirm your suspicion. Obviously large x9 files have a lot of images and hence we do not want to log everything. We only want to see the images that have failed EOFB testing.

Here are logging examples. Note that logging is only performed for EOFB failures, and then only up to the logging count that is specified in program options.

```
2014-01-10 06:29:32.805 [INFO] EOFB validation failed for record(22); end of image
segment(888888888888888888888FFFFFFFFFFFC00400) (id0)
2014-01-10 06:29:32.805 [INFO] EOFB validation failed for record(24); end of image
segment(FFFFFFFC1C1C1F111FFFFFFFFFFFFFFFFFFE0020) (id0)
```

Let's take a closer look at these two failures. Both are manufactured error situations but they do help to allow us to see actual image data and why a specific image has failed EOFB validation.

In the first example, the last four bytes of the image segment contains x'FFC00400". This segment clearly does not end in a valid EOFB bit string.

In the second example, the last four bytes of the image segment contains x'FFE00200". This segment clearly does not end in a valid EOFB bit string.

Summary

EOFB validation is a more complex x9 topic that is gathering a lot of interest within the x9 community. This type of validation can be done on a manual basis against one or two images. However, saying that this can be time consuming is a tremendous understatement of the effort that is required. You need automated tools to assist you in the process.

Appendix: X9Assist as a Testing Tool

About X9Ware LLC

X9Ware LLC provides extensive tools for users of the various x9.37 file specifications. Our product line extends from a free x9 viewer to tools that include validate, modify, make, generate, scrub, import, export, and provide a host of other x9 support and reporting functions. We have price competitive licensing options to meet the needs of any size organization. Our goal is to offer what we believe are some of the best tools in the industry and at the lowest possible cost.

X9Assist is our flagship product which incorporates a series of tools to support your x9 production support, testing, and development requirements. Specific examples are:

Tool	Usage
1) Make	Creates a new x9 file from a use case file which can be defined in either Excel or CSV format. A variety of fields can be provided via the use case file. Fields that are not present in the use case file can be assigned constant or random data based on your specific requirements.
2) Generate	Creates a new x9 file from a CSV file which can be obtained from various sources. The most typical usage of Generate is as the second step of the Make process (where the output from Make is the input to Generate).
3) Scrub	Obliterates fields within an existing x9 file by replacing values in selected fields with either random or use case data. The x9 data elements that can be altered include MICR line fields, amounts, item sequence numbers, endorsements, and file header records. Scrub also redraws images associated with an altered check after the new MICR fields are assigned, thus allowing the x9 data and the image to remain synchronized.
4) Clone	Clones bundles within existing cash letters to create large x9 files for performance and stress test purposes.
5) Merge	Merges cash letters from multiple independent files into a single, consolidated x9 file that can be used for testing purposes.

X9Assist EOFB Validation and Support

X9Assist uses TIFF rules XML to define when the EOF validation process is to be performed. Note that each defined x9 configuration can use an appropriate tiff rules XML definition. This allows you to define which x9 configurations will perform EOFB validations, and which will not.

```
<tiffControls>
   <endianFormat>littleEndian</endianFormat>
   <ifdOnWordBoundary>false</ifdOnWordBoundary>
   <requiredAscendingTags>true</requiredAscendingTags>
   <duplicateTagsAllowed>false</duplicateTagsAllowed>
   <privateTagsAllowed>true</privateTagsAllowed>
   <privateDuplicateTagsAllowed>false</privateDuplicateTagsAllowed>
   <multiStripAllowed>false</multiStripAllowed>
   <validateEOFB>true</validateEOFB>
   <imageMinWidth>4.100</imageMinWidth>
   <imageMaxWidth>10.500</imageMaxWidth>
   <imageMinHeight>1.752</imageMinHeight>
   <imageMaxHeight>5.700</imageMaxHeight>
</tiffControls>
```

In addition, the messages XML definition allows the severity of the EOFB error to be assigned (as either error, warn, info, ignore). Your x9 configurations can also optionally include a unique message XML definition, allowing you to automatically set the severity associated with these errors.

```
<error> <id>tiffImageSegmentDoesNotEndEOFB</id> <nbr>333</nbr> <sev>Warn</sev>
    <desc>tiff image segment does not end with EOFB</desc>
    </error>
```

Finally, you can use program options to indicate that you always want to perform EOFB validations. When this option is activated, EOFB validation will be performed for all x9 files regardless of the x9 rules that re being applied. Generated EOFB error severity will then be assigned as follows:

- If the current tiff rules require EOFB validation, then the error severity is assigned from the current message rules.
- If the current tiff rules do not require validation, then the error severity is set as informational.

For additional information on the capabilities of X9Assist, please contact lowell.huff@x9ware.com, or visit our website at www.x9ware.com.