

X9Ware-SDK User Guide

X9Ware SDK

Your x9.37+ACH+CPA005 support tools

Revision Date: 04/18/2024

Release R5.04

Copyright 2012 – 2024 X9Ware LLC

All enclosed information is proprietary to X9Ware LLC

X9Ware LLC

10753 Indian Head Industrial Blvd

St Louis, Missouri 63132-1101

(844) 937-1850

Email support@x9ware.com

Table of Contents

Overview.....	5
Upgrading from a Previous X9Ware-SDK Release.....	8
X9Ware-SDK Upgrade Considerations.....	8
X9Ware-SDK Resources.....	9
Code Upgrade Patterns.....	9
Try-With-Resources.....	19
Installation Tasks.....	20
JAR Requirements and ClassPath.....	21
SLF4J Logging.....	22
Logging Frameworks.....	22
X9SdkLogger.....	23
Linux Considerations.....	24
AWS Cloud Considerations.....	25
Submitting Problem Reports.....	26
Runtime Resources.....	27
Runtime Folders.....	27
System Folders.....	29
Logging During Startup.....	29
Explicitly Setting the System Folders.....	29
Explicit Assignment of the Launch Folder.....	30
Explicit Assignment of the Work Folder.....	30
Explicit Assignment of the Home Folder.....	30
X9Ware-SDK Fundamentals.....	31
X9Ware-SDK Initialization.....	33
License Keys.....	33
Sample Startup Code.....	33
Bind Configurations.....	34
X9.37 Configurations.....	35
ACH Configurations.....	35
X9Ware-SDK Shutdown.....	36
X9Ware-SDK Includes X9Utilities.....	37
Using X9Objects.....	41
Retrieving Fields within X9 Records.....	43
Modifying Fields within X9 Records.....	45
Credits And Trailer Totals.....	47
Using X9Writer.....	49
X9Ware-SDK Code Examples.....	50
Rules Overview.....	56

X9 Configurations.....	58
X9 Rules.....	61
X9 Rules – Base Specification Example.....	62
X9 Rules – Extension Specification Example.....	65
X9 Rules – X9Controls.....	70
X9 Rules – Basis.....	78
X9 Rules – X9Record.....	81
X9 Rules – Field.....	84
X9 Rules – Local Edits.....	88
X9 Rules – Cross Field Edits.....	89
X9 Rules – Date Range Validations.....	91
X9 Rules – Tables.....	91
X9 Rules – Tests.....	92
X9 Rules – POD Credit Tables.....	98
TIFF Rules.....	100
TIFF Rules – TIFF Controls.....	101
TIFF Rules – TIFF Edits.....	104
TIFF Rules – Mandatory TIFF Tags.....	106
TIFF Rules – TIFF Tag Descriptions.....	107
Gray Scale Image Support.....	108
X9 Messages.....	113
Message XML.....	113
System Messages.....	115
Override Messages.....	115
Full / Sparse / Plain.....	115
Message Patterns.....	116
Message Files in JAR versus File System.....	117
Message Pattern Reuse.....	117
Message Configurations.....	118
Using the Message and Configuration Editors.....	118
Using the X9Ware-SDK API to Insert Message Overrides.....	119
Using the X9Ware-SDK API to Insert Message Overrides from XML.....	120
Bitonal Image Thresholding.....	122
Bitonal Image Challenges.....	122
Bitonal Thresholding Techniques.....	122
MICR Line Format and Standards.....	126
MICR Line Standards.....	126
MICR Line Parsing.....	127

MICR Line Characters.....	127
MICR Line Fields.....	127
MICR Line Layout.....	128
MICR Line RegEx.....	129
Appendix: HeaderXml.....	132
Editing HeaderXml.....	132
HeaderXml as Written to the Log.....	133
X9 File Structure.....	134
Inclusion of Credits in Trailer Totals.....	134
HeaderXml Fields defined within the <info> group.....	134
HeaderXml Fields defined within the <fields> group.....	135
Appendix: X9 Record Types.....	153
Type 25 Check Detail Record.....	153
Type 26 Check Detail Addendum A Record.....	154
Type 27 Check Detail Addendum B Record.....	155
Type 28 Check Detail Addendum C Record.....	156
Type 31 Return Record.....	157
Type 32 Return Addendum A Record.....	159
Type 33 Return Addendum B Record.....	161
Type 34 Return Addendum C Record.....	161
Type 35 Return Addendum D Record.....	162
Type 61 Format (001) “Metavante”.....	163
Type 61 Format (002) “DSTU”.....	164
Type 61 Format (003) “x9.100-180”.....	165
Type 62 Format (000) “x9.100-187-2013”.....	166

Overview

The X9Ware-SDK is a full function Software Developer Kit (SDK) for Java programmers. It provides a wide of functions that are needed in support of X9.37, ACH, or CPA005 application development. Our design is to implement these various file formats (which we refer to as dialects) from a single SDK using a common API. Our goal has been to make this extensible to other dialects in the future, to allow our SDK to continue to meet the needs of the financial industry. The X9Ware-SDK is used extensively as the basis for all internally developed X9Ware products, and can similarly be used as the basis for your development work.

The X9Ware-SDK is 100% Java with minimal open source JARs. This simplifies our environment and reduces our software footprint. The X9Ware-SDK has a baseline requirement of JDK 1.8 on our desire to move to various functionality that was added in that release level including Lambda support.

We believe that the X9Ware-SDK has excellent capabilities and performance, when compared to any other tool you will find within the industry. Our validation engine (which supports both x9 and ach file formats) and associated rules are built into the X9Ware-SDK and can be leveraged by user implementations. The X9Ware-SDK contains many ancillary and advanced x9 functions for the developer such as TIFF image draw and IRD creation. The X9Ware-SDK is a proven product with customer implementations in Windows, Linux, and AIX. We have customer implementations using the Oracle JDK, various Open JDKs, as well as more complex environments such as Apache Tomcat, Spring, and WebSphere. The X9Ware-SDK supports full multi-threaded processing within a server environment.

The X9Ware-SDK allows inputs and outputs to be processed as files or streams (note that X9Utilities supports files only). This means that your application can have an x9.37 input file which can be easily exported to a CSV stream. Just as easily, your application can be completely stream based, without anything written to the file system. This is especially beneficial when running an application in the cloud (eg, within a docker container). The X9Ware-SDK has is fully self defined with all resource files embedded as internally defined resources. This implementation means that we have no dependencies on external files or folders. These resources include our internally defined components such as x9 rules, tiff rules, messages, packaged images, etc. Our technical design allows you to define these resources in external folders to override our JAR definition or add your own components as needed.

The X9Ware-SDK uses SLF4J to provide logging flexibility. An implementation is provided that is based on use of the JDK logger that can be replaced with LOG4J.

The X9Ware-SDK has been an evolving process and continues to be enhanced as we add new functions to our x9 tools. The philosophy at X9Ware has been to implement all primary x9 functions within the X9Ware-SDK and to then expose that functionality directly to our user base through our user base using such tools as X9Assist and X9Utilities. With this approach, you can essentially use the X9Ware-SDK to do the wide array of functions that are implemented in

X9Assist. For example, with the X9Ware-SDK you can: read, write, extract, and summary the content of input files; validate data and images; extract data and images; repair TIFF images; use templates to draw test items; draw image replacement documents (IRDs); and so forth.

The X9Ware-SDK is an evolving process and continues to be enhanced as we add new functions to our x9+ach tools. The philosophy at X9Ware has been to implement all primary functions within the X9Ware-SDK and to then expose and present that functionality using such tools as X9Assist and X9Utilities. With this approach, you can essentially use the X9Ware-SDK to perform a wide array of functions that are demonstrated in X9Assist. Our x9 rules engine is implemented within the X9Ware-SDK, which means that you can use it to both validate and create x9 files using any of our defined x9 rules configurations. You can also create your own customized x9 rules which will allow you to validate x9 files against your own x9 variants.

The X9Ware-SDK is not an afterthought. It is used internally by X9Assist to perform virtually all x9 related I/O functions. Most of the X9Assist tools are built into the X9Ware-SDK itself, so they are available to our SDK users. For example, x9 file validation and tiff image repair are complex functions that are available from the X9Ware-SDK. Consistent use of the X9Ware-SDK within X9Assist simplified the development of our product and we know that it can do the same for you. The X9Ware-SDK is also not a static product. Our philosophy of building upon the X9Ware-SDK will continue. This product will be expanded and improved as future X9Assist enhancements are implemented.

Our SDK includes Java classes that map the logical fields that exist within that x9 record type. This field level interface allows your application to work at the field level for both read and write (modify) operations. This field level interface also exists for the type 31 credit reconciliation record and is implemented in a manner that supports custom type 31 record formats per your own definition.

Examples of functions that can be performed by the X9Ware-SDK:

- Read/Write x9+ach files at both the record and field level
- Full multi-thread support to process multiple files concurrently
- Parse of input files (x9+ach) into record level objects that can be searched and modified
- File validation (x9+ach) at the record and image level
- X9 TIFF validation and repair
- Image rescaling and conversion
- Check and IRD image drawing
- Paid stamp for overlay onto back side check images
- Access individual images on both input and output
- Dynamically draw images to create x9 data for test systems (similar to Make)
- Dynamically draw IRDs (image replacement documents) including the endorsement chain and return reason
- File import/export to and from csv files
- Make/Generate/Scrub/Merge and other tools

- Extensive logging based on SLF4J (Simple Logging Facade) which is pluggable for log4j, logback, etc
- And other similar low level functions

The X9Ware-SDK has advantages over similar products offered by other vendors

- More functionality
- No timeouts or expiration dates built into the product
- No CPU or server level registration is required
- Once licensed, ongoing usage is based on maintenance renewals; we do not want your production applications aborting due to license key issues

Upgrading from a Previous X9Ware-SDK Release

The process of upgrading from an older X9Ware-SDK release to the most current release is normally very straight forward. We recommend that you incorporate the following steps in your plan:

X9Ware-SDK Upgrade Considerations

Important upgrade considerations are as follows:

- Consult with X9Ware to finalize the decision as to which X9Ware-SDK release you will install based on your project requirements and time line. In most situations, you will want to install the most recent “final” release of the X9Ware-SDK, which is a production release with general availability. However, dependent on your specific needs, you may also want to consider a “candidate” release which is under develop and will be released soon. This that be an advantageous decision if you either require capabilities that may only in that latest release, or if you require an enhancement that would be added explicitly by X9Ware as part of your project.
- Obtain the selected X9Ware-SDK build from X9Ware. If you have elected to use a candidate build, then add a task to your project plan to move to the final build of the chosen X9Ware-SDK release before you move to a product status.
- Review the JavaDoc for the current X9Ware-SDK release and especially in areas that are most used by your user applications. X9Ware has worked to minimize the API changes that we make to our X9Ware-SDK from release to release, but these will always exist. The potential for API differences will increase depending on the number of “skipped” X9Ware-SDK releases that are involved in your upgrade. The differences will be readily identifiable since your application program will get compiler syntax errors in those areas where there are API changes. It is then very important that you review the JavaDoc in the areas where you have API issues so you fully understand the new X9Ware-SDK capabilities in those functional areas where we have made changes. You can also review our current X9Ware-SDK Examples (which you can download from our website) and especially when you have used our examples as the basis for your development. Our examples will always be brought forward based on the current X9Ware-SDK API.
- Finalize a regression test plan. Regression testing should always consist of running your data through the “old” and “new” application environments. This testing must include automated file comparisons of the output files that are created since manual reviews are difficult and only scratch the surface of your program outputs. Output x9.37 files can be compared on an automated basis using tools such as X9Assist and HxD. Output CSV and image folders can be compared using tools such as WinMerge. Always include automated regression testing in your project plans.

X9Ware-SDK Resources

The various X9Ware-SDK resource files were stored externally for X9Ware-SDK release R3.03 and earlier. These resources include such files as Brand, Doc, Fonts, Images, Rules, and Xml. With our R3.04 release, these files have been moved to be internal to the X9Ware-SDK and no longer need to be defined and referenced in your file system at run time. This enhancement simplifies your JVM environment since these resource files no longer need to be pre-populated there and then included in your environment setups, backups, etc.

If you are moving from X9Ware-SDK R3.03 (or earlier) then you will want to remove these resource files from the new JVM/X9Ware-SDK environment you are building, since they are now embedded within the JAR. You can also remove your “launch folder” assignment process since that is similarly no longer required.

Code Upgrade Patterns

This is not a complete list since many small changes are made to our API that should be obvious from the current JavaDoc and our current X9Ware-SDK examples. Please work with us directly for any code changes that are problematic when upgrading from one release to another. We understand that this can become a challenge, and especially when numerous releases are being skipped as part of the upgrade. Your suggestions for this list are appreciated.

Release	Functional Change	New Code Patterns
R3.06	A new X9Factory class was created to simplify the process of creating x9 files on a field by field basis and as part of our x9.100-180 support. X9Factory will only attempt to populate those fields which actually exist in the current x9 standard.	See X9BuildX9 for complete examples as to how to use the new X9Factory class.
R3.06	X9SdkIO method renamed from writeFromCsvArray to writeFromArray.	<code>sdkIO.writeFromArray(sdkObject, csvArray);</code>
R3.06	X9SdkIO method renamed from startNewCsvOutputRecord to startNewCsvRecord.	<code>sdkIO.startNewCsvRecord(recordType, recordFormat);</code>
R3.06	X9ImportIE has been converted to a static class.	<code>X9ImageResults results = X9ImageIE.importImageToTiff(sdkBase, imageFile); byte[] byteArray = results.getByteArray();</code>
R3.06	X9ImageIE method importImage expanded into two separate methods: importImageToTiff and importImageToTiffWithRepair.	These methods load an image from an external file and provide the results as a TIFF byte array. If the image is in another format (PNG, JPG, etc) then it is converted from that format to TIFF.
R3.06	X9ImageIE method getBufferedImage expanded to include new method getBufferedImageWithRepair.	These methods load an image from an external file and provide the results as a BufferedImage. If the image is in another format (PNG, JPG, etc) then it is converted from that format to TIFF.

Release	Functional Change	New Code Patterns
R3.06	X9TiffRepair has been converted to a static class.	X9ImageResults results = X9ImageIE.importImage(sdkBase, imageFile); BufferedImage bi = results.getImage();
R3.06	X9FileIO can be used to write a byte array directly to an output file.	X9FileIO.writeToFile(byteArray, imageFile);
R3.06	X9SdkIO requires that images be attached to the X9SdkObject when using the write X9 from CSV functionality. See the current X9BuildX9 as an example.	When building your x9 file on a field by field basis. final byte[] imageByteArray = X9ImageLoader.getImageByteArray(imageFile); sdkObject.setCheckImage(tiffImage); sdkIO.addField(imageByteArray.length); // 52.18 sdkIO.addField(""); // 52.19 or when building your x9 file using X9Factory and constructing the entire csv as an array: sdkObject.setCheckImage(tiffImage); sdkIO.writeFromArray(sdkObject, csv);
R3.06	X9DrawTextLine renamed to X9TextLine.	textList.add(new X9TextLine("For Deposit Only", fontA30));
R3.06	X9AmountToWords changed to a static class.	final String amountString = X9AmountToWords.translateToWords(itemAmount);
R3.06	X9ImageLoader eliminated and replaced with X9FileIO.	final byte[] byteArray = X9FileIO.readFile(imageFile);
R3.07	X9AmountToWord line splitter moved to a new static class.	final List<String> amountList = X9LineSplitter.split(g2d, amountString, fieldSize, X9DrawItemFront.MINIMUM_WORDS_ON_FIRST_AMOUNT_LINE);
R3.07	The new "sequential" edit rule replaces our previous rules R26sequence, R28sequence, R32sequence, and R35sequence. The sequential edit rule validates that a field value when appearing in adjacent records of the same x9 record type. The edit validates that the value begins with one and is then incremented by one.	<field> <item>x26.02-p003-I001-mandatory-modifiable</item> <edit>n</edit> <edit>sequential</edit> <name>Check Detail Addendum A Record Number</name> </field>
R3.07	X9CsvReader no longer supports the skip header lines option since there are no absolute standards for their layout and they cannot be consistently identified.	If you know your CSV file contains a header attribute row, you can either write your own code to skip over that leading row or you can use the skip rows method to bypass that leading row.
R3.07	X9CsvReader and X9CsvWriter have been rewritten with numerous enhancements while remaining compatible with their previous functionality. These classes now	The CSV line buffer size is now determined internally. This has allowed us to eliminate the buffer size parameter within both X9SdkBase and the properties file. The X9CsvReader and X9CsvWriter constructors have been updated to longer require X9SdkBase since it was needed only to access the buffer size.

Release	Functional Change	New Code Patterns
	implement Closeable which allow them to take advantage of Java 1.7 try with resources.	X9CsvReader and X9CsvWriter are enhanced to support streams which provides more flexibility.
R3.07	X9JdkLogger close is deprecated and replaced with the new closeLog() method. This change was made since our direction is for all close() methods to implement Closeable and this was not the case for X9JdkLogger close().	X9JdkLogger.closeLog();
R3.07	X9TiffWriter has been changed to a static class to simplify usage.	final byte[] tiffByteArray = X9TiffWriter.createTiff(bw, imageDpi);
R3.07	X9ImageReader method readByteArray has been renamed to getImage.	final byte[] imageBuffer = x9imageReader.getImage(t52);
R3.07	X9GenerateFile method generateFile() now returns an error message (on failure) or null (on success).	final String generateErrorMessage = x9generateFile.generateFile();
R3.08	X9Object method isImageReplaced has been renamed to hasDirectlyAttachedImage.	If (x9o.hasDirectlyAttachedImage())
R3.08	X9Object method setReplacementImage has been renamed to setDirectlyAttachedImage.	x9o.setDirectlyAttachedImage(tiffArray);
R3.08	X9Object method getReplacementImage renamed to getDirectlyAttachedImage.	final byte[] tiffArray = x9o.getDirectlyAttachedImage();
R3.08	X9Reader has been renamed to X9Reader937.	final X937Reader x9reader937 = new X937Reader(sdkBase, x9streamReader);
R3.08	X9Reader937 method getX9DataLength has been renamed to getDataLength.	final int dataLength = x9reader937.getDataLength();
R3.08	X9Reader937 method getX9RecordType has been renamed to getRecordType.	final int recordType = x9reader937.getRecordType();
R3.08	X9SdkIO method getX9Reader has been renamed to getReader937.	final long fileLength = sdkIO.getReader937().getFileSize();
R3.09	JVM requirement upgraded from JRE 1.8 to JRE 1.9.	

Release	Functional Change	New Code Patterns
R3.09	X9SdkIOW has been removed (it was added with R3.07) with X9SdkIO itself now implementing Closeable.	try (final X9SdkIO sdkIO = sdk.getSdkIO()) {
R3.09	Class X9TrailerManager has been renamed to X9TrailerManager937, to be aligned with the new X9TrailerManagerAch.	X9TrailerManager is now an interface definition which is shared by x937 and ach. Example as follows: final X9TrailerManager x9trailerManager = new X9TrailerManager937(sdkBase);
R3.09	X9TrailerManager937 method getBundleTotals has been renamed to getBatchTotals to better describe the shared usage between x937 and ach.	final X9TrailerTotals totals = x9trailerManager.getBatchTotals();
R3.09	Field edit rule "Dateyyyymmdd" renamed to "yyyymmdd".	Change to be applied to appropriate x9 xml rules. Backward compatibility to old name is maintained.
R3.09	Field edit rule "Timehhmss" renamed to "hhmss".	Change to be applied to appropriate x9 xml rules. Backward compatibility to old name is maintained.
R3.09	Field edit rule "Timehhmm" renamed to "hhmm".	Change to be applied to appropriate x9 xml rules. Backward compatibility to old name is maintained.
R3.09	X9Validator constructor changed to accept file attributes as new class X9FileAttr and not the x9 reader.	final X9Validator x9validator = new X9Validator(sdkBase, sdkIO.getReader937().getFileAttributes());
R3.09	X9SdkIO has certain methods renamed to have a more generic name as part of our ACH implementation. This makes their method name apply equally well to x9 as well as to ach.	makeCsvFromX9 renamed to makeCsvFromInputRecord. getCsvFromX9 renamed to getCsvFromX9Object. makeX9FromCsv renamed to makeOutputRecordFromCsv. makeX9FromX9Object renamed to makeOutputRecord. makeX9FromSdkObject renamed to makeOutputRecord. createX9RecordFromCsvArray renamed to createOutputRecordFromCsvArray. readX9 renamed to readInputFile. writeX9 renamed to writeOutputFile. openX9InputFile renamed to openInputFile. openX9InputFileImageReader renamed to openImageReader. openX9OutputStream renamed to openOutputStream. openX9OutputFile renamed to openOutputFile. isX9OutputStreamOpen renamed to isOutputStreamOpen.

Release	Functional Change	New Code Patterns
		<p>closeX9InputFile renamed to closeInputFile.</p> <p>closeX9OutputFile renamed to closeOutputFile.</p> <p>closeX9ImageReader renamed to closeImageReader.</p>
R3.09	X9SdkObject has certain methods renamed to have a more generic name as part of our ACH implementation. This makes their method name apply equally well to x9 as well as to ach.	<p>getX9Data() renamed to getDataByteArray().</p> <p>getX9DataLength() renamed to getDataByteArrayLength().</p> <p>setX9Data() renamed to setDataByteArray().</p> <p>getX9AsciiRecord() to getAsciiRecord().</p> <p>getX9OutputRecord() renamed to getOutputRecord().</p> <p>getX9OutputRecordLength() renamed to getOutputRecordlength().</p> <p>buildX9FromData() renamed to buildOutputFromData().</p> <p>buildX9FromImage() renamed to buildOutputFromImage().</p> <p>buildX9FromDataAndImage() renamed to buildOutputFromDataAndImage().</p> <p>buildX9() renamed to buildOutput().</p>
R3.09	X9ImageReader renamed to X9RandomReader for clarity since it is used for a variety of random read purposes.	<pre>final X9RandomReader x9randomReader = new X9RandomReader();</pre>
R3.09	X9SdkBase method getImageReader() renamed to getRandomReader().	<pre>final X9RandomReader x9randomReader = sdkBase.getRandomReader();</pre>
R3.09	X9Generate937 and X9GenerateXml have changes which eliminate the xml fields that were associated with creation of invalid image scenarios. These were fields itemsPerTestCase and imageTests (which was a boolean character string which indicated the tests to be created).	<p>This previous implementation was difficult to use and was highly dependent on the indexes of each invalid image scenario and hence was also release dependent, since that those indexes could change from release to release. This was resolved with a new X9GenerateColumns field "invalidImageScenario" which is now specified at the item level and can be used to explicitly identify the invalid image scenario to be applied to each item as appropriate. This new design allows image test cases to be assigned on an item by item basis as needed and eliminates the index dependency.</p>
R3.09	Set launch folder moved from X9Properties to X9LaunchFolder.	<pre>X9LaunchFolder.setFolder(launchFolder);</pre>
R3.09	Set home folder moved from X9Properties to X9HomeFolder.	<pre>X9HomeFolder.setFolder(homeFolder);</pre>
R3.10	X9Util renamed to X9UtilMain as	<p>X9UtilMain implements closeable so it can be invoked using try</p>

Release	Functional Change	New Code Patterns
	part of our separate new batch functions such as X9Export, X9Merge, etc, and in conjunction with the new abstract class X9UtilBatch. The new X9UtilMain also allows utility functions to be invoked by SDK applications and facilitates the ability to not issue system exit.	with resources to ensure closed. Example is as follows: <pre>int exitStatus = EXIT_STATUS_ABORTED; try (final X9UtilMain x9utilMain = new X9UtilMain()) { exitStatus = x9utilMain.launch(args); } catch (final Throwable t) { LOGGER.error("throwable exception", t); } finally { System.exit(exitStatus); }</pre>
R3.10	X9Item renamed to X9Item937 with the addition of ACH support and the new X9ItemAch.	<pre>final X9Item937 x9item = new X9Item937(recordNumber, recordType, routing, onus, auxOnus, epc, X9Decimal.getAsAmount(amount), isn);</pre>
R3.10	X9CsvReader method getStringArray has been renamed to getNextIncomingStringArray since that name is a better description of the function performed.	<pre>/* * Call the csv reader to get the next csv input line. */ final String[] csvArray = getOpenedCsvReader().getNextIncomingStringArray();</pre>
R3.10	X9MicrOnUs (which parses the MICR OnUs string into component parts) has been changed to be an immutable class. The parse function has been moved into the constructor and a new getError() method is used to retrieve the validation status.	<pre>final X9MicrOnUs x9micrOnUs = new X9MicrOnUs(sdkBase, x9o.getMicrOnUs()); final X9Error x9error = x9micrOnUs.getError(); final String account = x9micrOnUs.getAccount(); final String pc = x9micrOnUs.getProcessControl();</pre>
R3.10	X9HeaderXml renamed to X9HeaderXml937.	<pre>private final X9HeaderXml937 x9headerXml937 = new X9HeaderXml937();</pre>
R3.12	X9MicrOnUs no longer requires "sdkBase" in the constructor. This reference is now only required when you specifically invoke the getError method.	<pre>final X9MicrOnUs x9micrOnUs = new X9MicrOnUs(micrOnUs); x9error = x9micrOnUs.getError(sdkBase);</pre>
R3.12	Static class X9RoutingValidate has been renamed to X9ValidateRouting as part of standardization.	<pre>final X9ModCheck x9modcheck = X9ValidateRouting.getModCheckInstance();</pre>
R3.12	The detailed xml files that exist within X9HeaderXml937 have been moved to an xml bean with associated attributes, which can be obtained using a getter method from X9HeaderXml937.	<pre>final X9HeaderXml937 x9headerXml937 = new X9HeaderXml937(); x9headerXml937.readHeaderDefinition(headerXmlFile); final X9HeaderAttr937 headerAttr = x9headerXml937.getAttr();</pre>
R3.12	The detailed xml files that exist within X9ScrubXml937 have been moved to an xml bean with associated attributes, which can be obtained using a getter method	<pre>final X9ScrubXml x9scrubXml = new X9ScrubXml(); x9scrubXml.loadScrubConfiguration(scrubXmlFile); final X9ScrubAttr scrubAttr = x9scrubXml.getAttr();</pre>

Release	Functional Change	New Code Patterns
	from X9ScrubXml937.	
R4.01	X9Object method getAmount() renamed to getRecordAmount().	This method now returns the specific amount that is set within this x9object.
R4.01	X9Object method getAmountAsLong() renamed to getRecordAmountAsLong().	This method now returns the specific amount that is set within this x9object.
R4.01	X9Object method setAmount() renamed to setRecordAmountAsLong().	x9o.setRecordAmount(BigDecimal.ZERO);
R4.01	X9SdkIO method readInputFile() renamed to readNext(), which is a better description, since it can process either files or streams.	sdkObject = sdkIO.readNext();
R4.01	X9SdkBase no longer extends X9SdkRecords. This change is part of our longer term SDK changes to isolate x9 specific definitions and make the SDK itself more generic. Elimination of X9SdkRecords generally has little to no impact to SDK based applications. It does mean that an x9 record field number must now be referenced through the new X9RecordFields class, instead of through the x9 field numbers that were generally available through the current X9SdkBase instance.	For example, this usage pattern has been eliminated. final int r52ImageDataFieldNumber = sdkBase.r52ImageData; Is now replaced by: final X9RecordFields x9recordFields = sdkBase.getRecordFields(); final int r52ImageDataFieldNumber = x9recordFields.r52ImageData;
R4.01	X9Object has two methods that can be very helpful when querying against the record number that is instantiated by the referenced x9object.	An example of querying if the record number is equal to: final boolean isImageRecord = x9o.isRecordType(X9.IMAGE_VIEW_DATA); An example of querying if the record number is not equal to: final boolean isNotImageRecord = x9o.isNotRecordType(X9.IMAGE_VIEW_DATA);
R4.01	X9StreamReader has been redesigned and replaced with X9FileReaderChannel and X9FileReaderStream, both of which implement X9FileReader.	try (X9FileReader x9fileReader = X9Reader.getNewReader(inputFile)) { }
R4.01	X9Field method getFieldNumber() renamed to getFieldIndex() to be more descriptive, since the returned value is zero based.	final int fieldIndex = x9field.getFieldIndex();
R4.04	Common tools within the SDK were moved to a separate Java project, which was needed to	An example is X9Exception, which has moved from package com.x9ware.core to com.x9ware.actions.

Release	Functional Change	New Code Patterns
	support our creation of E13B-OCR as a separate product. This resulted in a small number of package name changes.	These moves did not change the class APIs in any way. They were needed to ensure that we maintain unique package names across all of our internal Java projects.
R4.04	The SDK now includes a static license key which must be set as part of initialization. This license includes company name, client name, and expiration date.	The license key is static and as such will typically never be changed. Our design is to incorporate the license key directly into the SDK application code itself. This approach ensures that the key is always present.
R4.05	X9SdkBase methods getImageMode() and setImageMode() have been eliminated. They are replaced with an X9ImageMode parameter that has been added to several methods, which allow the value to be explicitly provided when needed.	<pre>while ((sdkObject = sdkIO.getNextCsvInputRecord(imageMode)) != null) { ... }</pre>
R4.05	The SDK has eliminated several X9ImageMode enum values that were used to indicate relative versus absolute image file names when x9.37 files were exported to CSV. These enums were replaced with a sdkIO setting that is assigned true when the generated image file names should be relative and false when absolute (the default action).	<pre>/* * Set image export option as either relative or absolute. */ sdkIO.setExportedFileNamesRelative(isImageExportRelative);</pre>
R4.05	X9CsvWriter has method addFieldExplicitly rename to adFieldAsTrimmed, which is a better description of the functionality.	<pre>for (final String csvValue : csvArray) { csvWriter.addFieldAsTrimmed(csvValue); }</pre>
R4.06	X9ExportFile constructor has been simplified for the typical case when there is only being one file exported. Setters are provided when X9ExportFile is being used for multiple files.	<pre>final X9ExportFile x9exportFile = new X9ExportFile(sdk, outputFile, x9exporter); x9exportFile.setTotalFileCount(inputFiles.size()); x9exportFile.setTotalFileByteCount(totalByteCount);</pre>
R4.06	X9ExportResults was renamed to X9ExportTotals as a better description of the functionality. The accumulators were only partially populated in earlier releases. They are now being more fully populated with totals being accumulated by X9TrailerManager.	<pre>final X9DecimalFormatter x9d = new X9DecimalFormatter(); String exportSummary = "file(" + inputFile + ") recordCount(" + x9d.formatLong(x9exportTotals.inputCount) + ") debits(" + x9d.formatLong(x9exportTotals.debitCount) + ") amount(" + x9d.formatDollarAmount(x9exportTotals.debitAmount) + ")";</pre>
R4.06	All method and attributes changed from gender to dialect.	<pre>if (sdkBase.isDialectX9()) { }</pre>

Release	Functional Change	New Code Patterns
R4.06	X9SdkIO method <code>openInputStream</code> has been renamed to <code>openInputReader</code> , as part of the implementation of mixed file and stream support in SdkIO.	<code>sdkIO.openInputReader(X9FileReader.getNewReader(inputStream));</code>
R4.06	X9Writer method <code>openAndBind</code> has been renamed to <code>bindAndOpenFile()</code> , as part of our stream support enhancements.	<code>x9writer.bindAndOpenToFile(x9outputFile, x9headerXml937);</code>
R4.06	X9FileReader method <code>getNewReader</code> has been renamed to <code>getNewChannelReader</code> as part of our stream implementation.	<code>try (X9FileReader x9fileReader = X9FileReader.getNewChannelReader(inputFile)) {</code> <code>}</code>
R4.06	X9FileReader method <code>getNewReader</code> has been renamed to <code>getNewStreamReader</code> as part of our stream implementation.	<code>try (final X9FileReader x9fileReader = X9FileReader.getNewStreamReader(inputStream)) {</code> <code>}</code>
R4.08	Various methods have a new X9Dialect enum provided as a parameter instead of that same value in string form. An example is X9SdkFactory <code>get()</code> .	<code>sdk = X9SdkFactory.getSdk(sdkBase, X9Dialect.X9);</code>
R4.08	Various methods have a new X9Dialect enum provided as a parameter instead of that same value in string form. An example is X9DialectFactory <code>getNewReader()</code> .	<code>final X9Reader x9reader = X9DialectFactory.getNewReader(sdkBase, X9Dialect.X9, x9fileReader, X9Reader.MAILBOX_NOT_ACCEPTED)) {</code>
R4.08	Various methods have a new X9Dialect enum provided as a parameter instead of that same value in string form. An example is X9ConfigManager <code>getConfigList()</code> .	<code>final String[] configurationList = X9ConfigManager.getConfigList(X9Dialect.X9);</code>
R4.08	X9ImageResults moved from <code>com.x9ware.imaging</code> to <code>com.x9ware.imageio</code> .	<code>import com.x9ware.imageio.X9ImageResults;</code>
R4.08	X9MessageManager moved from <code>com.x9ware.messaging</code> to <code>com.x9ware.base</code> .	<code>import com.x9ware.base.X9MessageManager;</code>
R4.08	Various messaging constants moved from X9MessageManager to X9Message.	<code>X9Message.ESEPARATOR</code> , <code>X9Message.PREFIX</code> , <code>X9Message.LINE_NUMBER</code> , etc.
R4.08	Method <code>createBlankImage</code> moved from X9DrawTools to X9BitonallImage.	<code>final X9BitonallImage bw = X9BitonallImage.createBlankImage(w, h);</code>
R4.09	The font sizes passed to X9DrawTools <code>drawPaidEndorsementStamp</code> were not properly sized, subject to the	The result is that smaller (and more logical) font size definitions must be provided to X9DrawTools <code>drawPaidEndorsementStamp</code> . More typically, these sizes will now be more like 10, 12, 14, etc. The previous sizes were

Release	Functional Change	New Code Patterns
	target image DPI.	incorrect and much larger.
R4.11	The static method used to create new image folders within the file system, as part of export with images, was moved from X9SdkObject to new class X9SdkImageBundle.	lastBundleFolderWritten = X9SdkImageBundle.createNewBundleFolderWhenNeeded(imageFolder, relativeFileName, lastBundleFolderWritten);
R5.01	Class X9UtilWorkResults has been eliminated, with results now returned using X9UtilWorkUnitList.	final X9UtilWorkUnitList workList = x9utilMain.launch(args1); exitStatus = workList.getExitStatus();
R5.01	All SDK API's have been updated to utilize LocalDate/LocalTime as a replacement for the older java.util.Date. LocalDate was introduced with Java 8, which is also the minimum JDK release for the SDK. LocalDate is immutable and provided a much improved API for developers. X9Ware was focused on making this transition as easy as possible. This was important, since we had to upgrade over 400k lines of code across all of our applications, so we needed the upgrade to be very straight forward as well.	A new X9LocalDate static class has been implemented which replaces our previous X9Date class. X9LocalDate contains the same methods as X9Date, with several additions to make things a bit more convenience. X9LocalDate no longer uses java.util.Calendar. Similarly, we have eliminated almost all usage of java.text.SimpleDateFormat within our code base. The SDK example programs have been upgraded. Please let us know if you have any questions. final LocalDate today = X9LocalDate.getCurrentDate(); final String todaysDate = X9LocalDate.formatDate(today, X9LocalDate.YYYY_MM_DD); final String messageId = "timestamp:" + X9LocalDate.formatDateTime(X9LocalDate.getDateTime(), "yyyyMMdd_Hhmmss_SSS");
R5.02	X9ExportInterface has a new method public boolean isExportImagesIrdFormat().	See SDK example X9ReadX9ExportCsv for usage.
R5.02	X9DecimalFormatter has been eliminated and replaced with new static class X9D. The new X9D class is thread safe.	final StringBuilder sb = new StringBuilder(); sb.append("Export completed; csv lines written(").append(X9D.formatLong(csvWriteCount));
R5.04	Static factory method X9Exception.abort has been replaced with X9Exception constructors that are fully source code compatible with the previous abort() method.	Changed: throw X9Exception.abort(ex); to: throw new X9Exception(ex);
R5.04	X9Field method getValueAsLowerCase() renamed to getValueAsIs(), as a better description of it's functionality.	Changed: final String newValue = x9field. getValueAsLowerCase (x9o); to final String newValue = x9field.getValueAsIs(x9o).
R5.04	Method createInputFileList() moved from X9FileUtils to new static class X9FileWalker.	final List<File> fileList = X9FileWalker.createInputFileList(workUnit.inputFile, includeSubFolders, inputFileExtensions, SKIP_INTERVAL_ZERO, isLoggingEnabled);

Release	Functional Change	New Code Patterns

Try-With-Resources

The Java try-with-resources construct is supported beginning with our SDK R3.07 release. This enhancement uses the improved exception handling that was implemented with Java 1.7 that will automatically and correctly close resources that are used within a try-catch block. X9Ware has implemented the Closeable interface in our following classes:

- X9SdkIO
- X9StreamReader
- X9Reader
- X9Writer
- X9CsvReader
- X9CsvWriter

Try-with-sources has been used in our various X9Ware-SDK examples, so please reference them as documentation on how to use this facility.

X9SdkIO implements Closeable so it can now automatically close all input files that have been opened when invoked within a try-with-resources block. In order to provide backward compatibility, it still provides separate close methods that can be invoked for each of the associated input and output files. The X9SdkIO close() method has been implemented as follows:

```

/**
 * Close all is a convenience method in support of Closeable and try-with-resources. We close
 * all files that are currently open for this X9SdkIO instance; all file related processing must
 * be completed when we are invoked. This sdkIO method can be invoked directly or indirectly as
 * part of try-with-resources. Note that sdkIO statistics can be retrieved within an associated
 * try block since no additional IO will be performed. Although close all is convenient, it is
 * not applicable to all possible logic flows. An example is a file merge where one output file
 * is opened but where separate input files will be opened and closed as the merge progresses.
 * In that situation, close all does not apply and our lower level close routines are used
 * instead. Hence those routines remain as public to be invoked externally.
 */
@Override
final public void close() {
    closeInputFile();
    closeOutputFile();
    closeImageReader();
    closeCsvInputFile();
    closeCsvOutputFile();
}

```

Installation Tasks

SDK installation consists of the following basic tasks:

1. Read this guide for a full understanding of the X9Ware-SDK.
2. Review the provided Java examples to get a better idea of X9Ware-SDK application design and to assist in your planning on how to use the X9Ware-SDK within your environment.
3. Review the distribution materials provided within the X9Ware-SDK zip distribution package.
4. Build your JVM environment that will host the X9Ware-SDK and your Java application; we require JRE 1.8 or higher.
5. Establish your desired logging subsystem which can be built on LOG4J (or others) based on the logging systems that are supported by the SLF4J facade. Required JARs must be added to your JVM environment. Review the logging topic for more information.
6. (Optionally) add the X9Ware external resource libraries to your JVM environment. When doing this, the high level folder location must be assigned at run time using `X9LaunchFolder.setFolder()`. These libraries are not part of the standard X9Ware-SDK distribution but can be requested from X9Ware, or they can be used directly as distributed from the associated distribution level of our X9Assist product. This step is typically not needed, and would be done only when your environment requires a customized version of resources versus what is packaged within the standard JAR.
7. Ensure you have an adequate JVM heap size set for your application based on your environmental requirements and anticipated file sizes. The minimum heap size needed by our X9Ware-SDK is 100MB and you should increase from that size as needed.
8. If you have startup problems, please provide the log and the issue description to X9Ware for our research and resolution.
9. Use an X9Ware provided Java sample or your own written Java program to perform initial testing of your JVM environment. If you have problems, review the system log which will help to identify the issue that has been encountered. If necessary, follow our problem reporting topic to provide information to X9Ware to get your issue resolved.

JAR Requirements and ClassPath

X9Ware has worked to minimize the inclusion of open source and third party products within the X9Ware-SDK. This continuous effort results in several benefits including a reduced software footprint, fewer software dependencies, a reduced potential for release level conflicts across multiple applications when running within a shared JVM, and reduced complexity.

The required jars needed by the X9Ware-SDK are as follows:

Product and Release Level	Requirement	Purpose and Comments
SLF4J: we are using 1.7.30 but any recent release should be functionally acceptable.	Mandatory	Per the SLF4J web site: The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks, allowing the end user to plug in the desired logging framework at <i>deployment</i> time.
SLF4J plugin: Logging environment plugin of your choosing which must match the SL4J API JAR release level included in your class path.	Mandatory	Available logging frameworks are Log4J, LogBack, Logback, Java Util Logging, Simple, and None.
Apache Commons Lang3: we internally use the 3.5 release of this product.	Mandatory	Per the Apache Commons web site: Provides highly reusable static utility methods with a wide range of functionality.
JAXB: which was included with the JRE through Java 8 and either deprecated or removed with subsequent releases.	Required when using Java 9 or higher.	Per the Jaxb website: The Java Architecture for XML Binding (JAXB) provides an API and tools that automate the mapping between XML documents and Java objects. We are currently using JAXB 2.4.0. Refer to our current distribution “x9wareLib” for the actual jars being used. Specifically, the requirements are: jaxb api and runtime; istack tools and runtime; javax activation.

The X9Ware-SDK jar as packaged and distributed does not include a Java “.classpath” file. All class path requirements must be fulfilled by your “-cp” parameters.

Our recommendation is to consider creating a sub-folder with all X9Ware required jars and then using a wildcard to include the contents of that folder. For example, “-cp x9wareLib/*” can be used to generically include all jars in a sub-folder and simplifies ongoing maintenance as the list might change.

SLF4J Logging

Logging functionality is absolutely critical to every application. X9Ware realizes that every X9Ware-SDK user will have their own preferred logging implementations and standard processes that are critical to their environments. This may include specific logging frameworks, formatting rules, exits, and tools which implement automated cutoffs and archival.

X9Ware has utilized the SLF4J interface to avoid imposing a required single logging framework. Using SLF4J, there is flexibility to choose your logging environment at deployment time by inserting the corresponding SLF4J binding on the class path. This decision may then be changed at any time by replacing this binding with another on the class path and restarting the application. This SLF4J design approach has proven to be simple and robust, and has evolved over time to increase flexibility.

As of SLF4J version 1.6.0, if no binding is found on the class path, then the SLF4J API will default to a no-operation implementation and will then discard all log requests. Without a valid binding, SLF4J emits a single warning message about the absence and then discards all log requests without further protest. This is not acceptable, since you will need log output to monitor execution and provide the input you will need on research and problem resolution.

Logging Frameworks

SLF4J supports various logging frameworks. The SLF4J distribution ships with various jar files that are referred to as "SLF4J bindings", where each binding corresponding to a supported logging framework. You will need to review the SLF4J online documentation and use that to determine the jars that will be needed for your specific environment. The various SLF4J frameworks are as follows:

slf4j-jdk14-x-x-x.jar	Binding for java.util.logging, also commonly referred to as JDK logging.
slf4j-log4j-x-x-x.jar	Binding for log4j version, a widely used logging framework.
slf4j-simple-x-x-x.jar	Binding for Simple implementation, which outputs all events to System.err. Only messages of level INFO and higher are printed. This binding may be useful in the context of small applications.
slf4j-jcl-x-x-x.jar	Binding for Jakarta Commons Logging. This binding will delegate all SLF4J logging to JCL.
logback-classic-x-x-x.jar	The native implementation There are also SLF4J bindings external to the SLF4J project, e.g. logback which implements SLF4J natively. Logback's ch.qos.logback.classic.Logger class is a direct implementation of SLF4J's org.slf4j.Logger interface. Thus, using SLF4J in conjunction with logback involves strictly zero memory and computational overhead.

slf4j-nop-x-x-x.jar	Binding for NOP, which silently discards all logging.
---------------------	---

X9Ware defaults to using the JDK logger. In this situation, X9Ware will dynamically create the configuration files. Only two additional jars must be added via the class path:

```
slf4j-api-xx.jar           (X9Ware currently using 1.7.30)
slf4j-jdk14-xx.jar        “
```

Another example is using Log4j2, where the following jars will be needed on the class path:

```
slf4j-api-xx.jar           (X9Ware currently using 1.7.30)
log4j-api-xx.jar          (X9Ware currently using 2.17)
log4j-core-xx.jar         “
log4j-slf4j-impl-x.jar    “
```

Configuration of Log4j2 can be accomplished in one of several ways. Refer to the Log4j2 for more information. The configuration options are:

- Through a configuration file written in XML, JSON, YAML, or properties format.
- Programmatically, by creating a ConfigurationFactory and Configuration implementation.
- Programmatically, by calling the APIs exposed in the Configuration interface to add components to the default configuration.
- Programmatically, by calling methods on the internal Logger class.

X9SdkLogger

X9SdkLogger is our internal class that may be used to used to initiate logging when the JDK logger is to be utilized. X9JdkLogger allows you to explicitly define the folder location to be used for all log files. When omitted, logging will be done to the “log” folder within the system work folder.

```
final String LOGGING_FOLDER_SWITCH = "log";
final File[] files = X9CommandLine.parse(args);
final String logFolder;
if (X9CommandLine.isSwitchSet(LOGGING_FOLDER_SWITCH)) {
    logFolder = X9CommandLine.getSwitchValue(LOGGING_FOLDER_SWITCH);
    X9JdkLogger.initialize(new File(logFolder));
} else {
    logFolder = "";
    X9JdkLogger.initialize();
}
```

Linux Considerations

Setting up the X9Ware-SDK for Linux is essentially the same for either Windows or Linux installations.

We have seen some Linux systems throw the following error during startup:

```
Exception in thread "main" java.lang.InternalError: Can't connect to X11 window server ....
  at sun.awt.X11GraphicsEnvironment.initDisplay(Native Method)
  at sun.awt.X11GraphicsEnvironment.access$200(X11GraphicsEnvironment.java:65)
  at sun.awt.X11GraphicsEnvironment$1.run(X11GraphicsEnvironment.java:110)
  at java.security.AccessController.doPrivileged(Native Method)
  at sun.awt.X11GraphicsEnvironment.<clinit>(X11GraphicsEnvironment.java:74)
  .....
  .....
```

This can be resolved in one of several ways:

- Specify the `-Djava.awt.headless=true` parameter at startup time
- Or, add the following at the very beginning of your X9Ware-SDK application program

```
X9SdkRoot.setHeadless();
```


AWS Cloud Considerations

We have numerous customers running the X9Ware-SDK with the AWS (Amazon Web Services) cloud platform.

Based on customer feedback, the easiest way to accomplish this is to use Elastic Beanstalk.

Per the Amazon website, AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with **Java**, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS. Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring. At the same time, you retain full control over the AWS resources powering your application and can access the underlying resources at any time. There is no additional charge for Elastic Beanstalk, where it is part of the core AWS product.

Our customers who have deployed X9Ware-SDK applications using Elastic Beanstalk have indicated that they did not have any issues arise during their development, testing, and deployment. The automation that was provided greatly simplified the overall process.

Elastic Beanstalk provides services in the following areas:

- Simplified setup and installation
- Developer productivity
- Resource Control
- Automated scaling

Submitting Problem Reports

X9Ware has worked hard to provide the best possible product to our customers. However, problems can and will happen. Many are unique to the client's technical environment or issues that are specific to the installation. Issues can arise to your use of specific X9Ware-SDK functions. If a problem arises, X9Ware will work with you to resolve the problem as quickly as possible.

To work on a problem, we request the following information be provided:

- A brief description of what your application is trying to accomplish.
- The system log from the failure. A new log is created for each X9Ware-SDK execution. The logs are written to the system work folder unless overridden during start-up by your X9Ware-SDK application itself. The individual logs are time stamped so please provide the log that goes along with your failure.
- Any supporting information that may be helpful.

Runtime Resources

The X9Ware-SDK requires a series of resources which contain components that are associated with x9 file formats, x9 validation, tiff validation, image templates, internal fonts, and so forth. These resources can be located and accessed using one of the following techniques:

- 1) Use the resources which are embedded within the JAR. The X9Ware-SDK has all needed components as self defined and internally embedded within the “/resources” folder within the JAR. You can use a ZIP tool to take a detailed look at these files and folders. Components are located and loaded from the JAR as needed at execution time. This approach is recommended for all X9Ware-SDK users whenever possible due to the simplicity.
- 2) Modify or extend the resources within the JAR. This is an advanced topic and is typically used when you want to modify or extend our distribution components. In this case you would use a ZIP tool to unzip the X9Ware-SDK JAR, make your modifications as needed to the created folder structure, and then re-zip those folders to recreate your JAR. This approach is recommended because the JAR still contains all code and resources which are needed for your environment. All very beneficial.
- 3) Finally, define resources in your file system in either the home or launch folders. This is an advanced topic and is typically used when you want to modify or extend our distribution components and you do not want to apply those updates within the JAR. In this case, you can use the resources as they would be obtained from X9Ware as the basis for your modifications. You can then position these folders within your file system, in either the home or launch folders. Be sure to use matching X9Ware-SDK and resource build levels as you populate these folders, since the content and structure of these components may change from release to release.

Runtime Folders

Folder	Description
fonts	Contains various fonts which are used by the X9Ware tools.
images	Contains various image templates which are used by the X9Ware tools.
invalidImages	Contains various invalid image test cases which are pre-populated by X9Ware for invalid image scenario testing.
log	Contains log files when you use the X9JdkLogger as provided by X9Ware. This folder is used by our JDK logging routines but would not be used for example by your LOG4J environment.
properties	Contains the properties file that is internally defined and to be used for the X9Ware-SDK environment.
rules	Contains the rules that define your x9 configurations. Sub-folders are: <ul style="list-style-type: none"> • messages • tables

Folder	Description
	<ul style="list-style-type: none"> • tiffrules • x9rules
xml	Contains XML files and folders. The following are used during environment initialization: <ul style="list-style-type: none"> • checkFormats.xml • config.xml • map.xml • modCheck.xml • routing.xml

Informational messages are included in the log which identify the location which has been determined for these resource files.

This example shows all resource folders being loaded from the JAR:

```

2015-10-04 10:51:48.601 [INFO] folder(fonts) location(jar/resources/fonts)
(com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)
2015-10-04 10:51:48.602 [INFO] folder(images) location(jar/resources/images)
(com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)
2015-10-04 10:51:48.602 [INFO] folder(invalidImages) location(jar/resources/invalidImages)
(com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)
2015-10-04 10:51:48.603 [INFO] folder(rules) location(jar/resources/rules)
(com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)
2015-10-04 10:51:48.603 [INFO] folder(xml) location(jar/resources/xml)
(com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)

```

This example shows the fonts and invalid images folders being loaded from the JAR while other resources have been located in external folders:

```

2015-10-04 10:45:21.372 [INFO] folder(fonts) location(jar/resources/fonts)
(com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)
2015-10-04 10:45:21.372 [INFO] folder(images) location(C:\Users\X9Ware2\Documents\
x9_assist\images) (com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)
2015-10-04 10:45:21.373 [INFO] folder(invalidImages) location(jar/resources/invalidImages)
(com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)
2015-10-04 10:45:21.373 [INFO] folder(rules) location(C:\Users\X9Ware2\X9WareDrive\
X9WareGitRepository\x9Assist\rules) (com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)
2015-10-04 10:45:21.374 [INFO] folder(xml) location(C:\Users\X9Ware2\Documents\x9_assist\
xml) (com.x9ware.base.X9SdkRoot.logStartupEnvironment:112)

```

System Folders

The X9Ware-SDK has several core folder locations which are assigned on a default basis during startup and can be overridden as needed. These folders are as follows:

Folder	Folder Usage
Launch Folder	The launch folder represents the file folder location from which the X9Ware-SDK has been installed and launched. This folder was important for early versions of the X9Ware-SDK but is no longer critical since resource files are now embedded within the X9Ware-SDK and not referenced externally. The launch folder is read-only. The X9Ware-SDK will never write to the launch folder. The only need for the launch folder would be to load modified rules files (etc) should that be a user requirement. It is instead recommended that those files be packaged within X9Ware-SDK jar and that they are not loaded from the file system.
Work Folder	The work folder represents a file folder where the X9Ware-SDK can locate various system modifiable files and folders that are read and written during run time. The work folder requires write privileges, which is reason for its existence and logical separation from the launch folder. For example, if you are using the JDK logger, then the LOG folder will be assigned within the work folder. Another possible use of the work folder is to host temp files, which can be used for the creation of various intermediate files by certain X9Ware-SDK functions.
Home Folder	The home folder represents a file folder where the X9Ware-SDK can locate various user modifiable files and folders that are read and written during run time. The home folder is used by the X9Ware X9Assist product but is typically not used by other X9Ware-SDK based applications.

Logging During Startup

The system folder locations are written to the system log during startup. This logging includes their location as well as the assignment method. An example is as follows:

```
2017-02-13 09:00:35.345 [INFO] properties defaulted
2017-02-13 09:00:35.346 [INFO] launchFolder set from absolute path(C:\xx\xx\xx)
2017-02-13 09:00:35.347 [INFO] homeFolder set from FileSystemView( C:\xx\xx\xx)
2017-02-13 09:00:35.347 [INFO] workFolder set from AppData(C:\xx\xx\xx)
```

Explicitly Setting the System Folders

The system folder locations can be explicitly assigned when they are to be overridden from default locations. These overrides must be assigned before the X9Ware-SDK instance is allocated. This is typically an advanced topic and is not required by most X9Ware-SDK installations. If you do not explicitly assign these folder locations, then the X9Ware-SDK will utilize default logic to determine their location. The default assignment logic has been developed and fine tuned for our X9Assist and X9Utilities products, which may (or may not) be appropriate for your specific

environment. It is suggested that you thoroughly test these assignments and ensure that you have a perfected strategy for your launch and home folder assignments.

Explicit Assignment of the Launch Folder

The launch folder must be assigned first (before the home folder). As noted above, the X9Ware-SDK default is to assign a shared folder location for the launch and home folders. With that default in place, you then will only need to specify the launch folder, with that assignment then also be used for the home folder. You can explicitly assign the launch folder as follows:

```
File launchFolder = new File("c:/yourApplication/runTime/x9sdk");
X9LaunchFolder.setFolderFolder(launchFolder);
```

Explicit Assignment of the Work Folder

The work folder may be optionally assigned after the launch folder location has been specified but only in specific conditions as needed by your application:

```
File workFolder = new File("c:/home/x9ware/x9sdkWork/");
X9Properties.setWorkFolder(workFolder);
```

Explicit Assignment of the Home Folder

The home folder may be optionally assigned after the launch folder location has been specified but only in specific conditions as needed by your application:

```
File homeFolder = new File("c:/home/x9ware/x9sdkHome/");
X9HomeFolder.setFolder(homeFolder);
```

X9Ware-SDK Fundamentals

The following core classes exist with the X9Ware-SDK:

Class	Description
X9SdkRoot	X9SdkRoot is a static class that hosts system wide information within the SDK. Most functions within X9SdkRoot are used during your application initialization process and before your first X9SdkBase instance is created.
X9SdkBase	A new X9SdkBase instance is created for each SDK thread that is being processed within your application. Most applications will run with a single X9SdkBase instance. X9SdkRoot assignments must be made before your first X9SdkBase instance is allocated.
X9Sdk	A new X9Sdk instance is created for each independent processing function within your application. X9SdkBase can own one or more X9Sdk instances. Most applications will use a single X9Sdk instance but that is not an absolute requirement. For example, if you application needs to read two separate x9 files at the same time, you would want to allocate a single X9SdkBase and then two X9Sdk instances, where each of these would be used to read a single x9 file.
X9SdkIO	X9SdkIO contains IO related fields for the current X9Sdk instance. Each X9Sdk instance has a single X9SdkIO instance which is allocated in the X9Sdk constructor. An X9SdkIO instance can accommodate a single logical grouping of input and output files. For example, a single X9SdkIO instance can support x9 input and x9 output, CSV input and x9 output, or x9 input and CSV output.
X9SdkObject	X9SdkObjects are used by X9SdkIO to represent a single x9 record. This object is created for every possible x9 record type (not just checks). In addition to x9 record and field data, there are also fields with more limited usage that apply only to checks. This includes the check amount and image based attributes such as the image itself (in one of several forms) and an indicator as to whether the image represents the front or back of the check. x9 data can be initially presented to X9SdkObject in one of several forms. The first option is to provide the data from an external source which can be either an x9 or csv file. The second option is to provide the x9 data from an internally created csv array, which represents the individual fields that are to be ultimately used to create an x9 record. The final option is to build the x9 data on a record by record and field by field basis.
X9Object	X9Object is the internal representation of each x9 record type and is core functionality within the SDK and our X9Assist application. X9Objects can be stored into an array list using methods provided by X9ObjectManager. Note that x9 data is stored into X9Objects but the tiff images are only optionally stored, which can dramatically reduce the amount of memory required by the x9objects for an entire x9 file. Random access to tiff images is provided using X9ImageReader to obtain images when not stored. X9Object provides a large number of fields that provide record level information such as record number, record type, and the raw

Class	Description
	data data for this x9 record. There are a large number of methods that provide access to data at the field level and also allow you to walk all x9objects in a forward or backward direction.

X9Ware-SDK Initialization

Every application environment has possible unique features, so it is difficult to present a boiler plate that shows every possible X9Ware-SDK initiation and termination requirement that might be applicable to every technical environment.

Our goal is here to highlight a sample X9Ware-SDK initialization process, which should be generally appropriate for many environments.

Remember that X9SdkBase is normally only created once for your application, but would instead be created once per thread when you are processing within a multi-threaded environment. We do recommend engaging X9Ware Consulting Services for complex environments, to ensure your success.

License Keys

The X9Ware-SDK is specifically licensed to a company-contact, with the intent to identify accountability and ownership. X9Ware provides an encrypted licensing XML document to each customer which is a static string that must be set as part of initialization. Each license includes an expiration date, where a value of “12/31/9999” is assigned for perpetual licenses.

See the highlighted code below that sets the license key for the current environment.

Sample Startup Code

This sample code illustrates high level code structure for a typical x9 application:

```
/*
 * Open the JDK logger.
 */
X9JdkLogger.initialize();

/*
 * Invoke your application.
 */
try {
    /*
     * Initialize the environment.
     */
    X9BuildAttr.setSdkProductLicense(licenseXmlDocument);
    X9SdkRoot.LogStartupEnvironment("YourProgramName");
    X9SdkRoot.LoadXmlConfigurationFiles();

    sdkBase = new X9SdkBase();

    /*
     * Set the x9 configuration which defines the x9 rules.
     */
    if (!sdkBase.bindConfiguration(X9.X9_37_CONFIG_NAME)) {
```

```

        throw new X9Exception("bind unsuccessful");
    }

    /*
     * Automatically calculate trailer totals.
     */
    sdkBase.setRepairTrailers(true);

    /*
     * Run your application.
     */
    process();

} catch (Exception ex) {
    LOGGER.error("exception", ex);

} finally {
    X9SdkRoot.shutdown();
    X9JdkLogger.close();
    System.exit(0);
}

```

Please be advised that this is just a sample. Specific coding will be dependent on your technical environment and your specific needs.

Bind Configurations

The X9Ware-SDK must bind to a configuration during the initialization process. A configuration defines the records, fields, and validation rules which are applied. There are a variety of predefined configurations, where each represents a commonly used set of rules used through the industry today.

```

/*
 * Standard configurations.
 */
public static final String ACH_NACHA_2013_CONFIG = "nacha-2013";
public static final String ACH_CORE_VALIDATIONS_CONFIG = "nacha-core-
validations";
public static final String ACH_NO_VALIDATIONS_CONFIG = "nacha-no-validations";
public static final String X9_DSTU_NO_VALIDATIONS_CONFIG =
    "x9.dstu-no-field-validations";

public static final String X9_37_CONFIG = "x9.37";
public static final String X9_100_180_2006_CONFIG = "x9.100-180-2006";
public static final String X9_100_180_2013_CONFIG = "x9.100-180-2013";
public static final String X9_100_187_2008_CONFIG = "x9.100-187-2008";
public static final String X9_100_187_2013_CONFIG = "x9.100-187-2013";
public static final String X9_100_187_2016_CONFIG = "x9.100-187-2016";
public static final String X9_100_187_UCD_2008_CONFIG = "x9.100-187_UCD-2008";
public static final String X9_100_187_UCD_2013_CONFIG = "x9.100-187_UCD-2013";
public static final String X9_100_187_UCD_2016_CONFIG = "x9.100-187_UCD-2016";
public static final String X9_100_187_UCD_2018_CONFIG = "x9.100-187_UCD-2018";
public static final String X9_CPA_015_CONFIG = "x9.CPA_015";
public static final String X9_EC_ACH_CONFIG = "x9.EcAch";

```

```
public static final String X9_EEX_CONFIG = "x9.Eex";
public static final String X9_FRB_CONFIG = "x9.Frb";
public static final String X9_SVPCO_CONFIG = "x9.SvpCo";
public static final String X9_VIEWPOINTE_CONFIG = "x9.Viewpointe";
```

The most basic configuration for x9.37 is: X9_37_CONFIG.

The most basic configuration for x9.37 is: ACH_NACHA_2013_CONFIG.

In order to become familiar with the various configurations, we recommend that you use either X9Validator or X9Assist to display the same physical file while flipping the configuration from one setting to another. This will allow you to both see and experience the validations that are applied by each of these, and help to provide insight into their differences.

X9.37 Configurations

The X9.37 environment is known to have numerous file specifications that have evolved since their introduction in 2003. These specifications have extreme differences in both the fields that are present as well as the validation rules to be applied.

The x9.100-180 specifications are probably the wild card due to their late introduction and the large number of core differences that are present within this standard. Due to a number of factors and their associated complexity, the x9.100-180 specification is used infrequently within the industry today.

The x9.37 specifications include high-level controls that identify their core attributes. This includes the use of field zero (which is the four byte prefix on the front of each data record) and character set encoding (EBCDIC versus ASCII).

ACH Configurations

ACH has an anomaly where fields can be defined as either “mandatory”, “required”, or “optional”. There is a very fuzzy difference between mandatory and required. The presence of mandatory fields is absolute, while the presence of required fields is deferred to the receiver. The reality is that most fields that are defined as “required” will be needed by the receiver. For example, account number is defined as required, when in reality a transaction is incomplete without this core data element.

ACH_NACHA_2013_CONFIG represents the NACHA standard including the field level definition of mandatory versus required. When used in its most basic state, ACH_NACHA_2013_CONFIG will treat required fields as optional, due to the nature and spirit of the specification. However, also note that ACH_NACHA_2013_CONFIG can be dynamically configured to treat all required fields as mandatory. This is accomplished by setting X9Options.requiredFieldsAreMandatory = true.

ACH_CORE_VALIDATIONS_CONFIG is an extension of ACH_NACHA_2013_CONFIG which overrides the most commonly needed fields as mandatory. Obviously, this is a somewhat arbitrary decision, due to the gray areas that the specification itself was attempting to avoid.

X9Ware-SDK Shutdown

There are several considerations for your X9Ware-SDK shutdown sequence:

- All of your application processes should be completed.
- X9ImageReader should be closed if you have it opened against an x9 file.
- X9Ware-SDK shutdown should be performed.
- Your logging environment should be appropriately closed.

X9Ware-SDK shutdown can be performed as follows:

```
/*
 * Shutdown the sdk.
 */
X9SdkRoot.shutdown();
```

This X9Ware-SDK shutdown process consists of the following activities:

- Final X9FontCache statistics are logged.
- Final X9FontManager statistics are logged.
- Any open X9ImageReader instances are closed.
- X9ThreadPool is closed which terminates all threads.

A shutdown parameter allows you to indicate if shutdown statistics should be logged, which defaults to enabled (true).

Shutdown will not close the current SLF4J logging environment, which you may need to do after your shutdown has been completed. The JDK logger that is implemented by X9Ware can be closed as follows:

```
/*
 * Close the logging environment.
 */
X9JdkLogger.close();
```

X9Ware-SDK Includes X9Utilities

The X9Ware SDK incorporates X9Utilities within its overall framework, and specifically from the perspective of a Java application. SDK customers can invoke X9Utilities in one of two ways:

- From a Java program that you develop, which invokes the X9Utilities classes within the SDK, using the SDK jar itself. When used in this manner, you are invoking utility functions from your own application program. This allows you to incorporate these proven functions directly into your application, but also requires knowledge of the overall operation of X9Utilities and how you can integrate it into your application. We have designed X9UtilMain in such a manner to make this possible.
- Using a separately provided X9Utilities runtime jar, which is provided on our website download page (in addition to the SDK download package). When used in this manner, you are invoking X9Utilities in the same way as other customers that have purchased our X9Utilities standalone product, and have not purchased the SDK. To do this, you will need to download and install X9Utilities and use a separately provided SDK encrypted license key (elicense.txt), which must be stored into the 'license' folder. This elicense.txt file is provided to you at the time of your SDK purchase and is a perpetual key. Please contact us if you need the elicense.txt file regenerated for you. Note that we do not provide the Windows based installer for X9Utilities and only the JAR package.

All classes that comprise the X9Utilities product are encapsulated within the SDK itself, and are exposed as part of the public SDK API. Consequently, a Java program can utilize these classes **programmatically** (by that we mean from a Java program), either as individual utility functions, or by launching them through the main utility class.

The X9Utilities main class allows you to implement your preferred logging environment, through the SLF4J logging facade. For instance, you can use Log4J or LogBack, and not be tied to the JDK logger as is utilized by our distributed batch utilities product. Moreover, you have the flexibility to invoke X9Utilities multiple times within a single run unit or thread, enabling you to open the log, invoke one or more utility functions, monitor exit status, and make decisions regarding how to proceed with processing. For example, with this type of design, you could invoke a merge followed by an export. Using these proven tools, you can easily integrate your application code with one or more utility functions to build a more comprehensive solution tailored to your needs.

However, despite these capabilities, it's important to note a specific limitation: you cannot execute the X9Utilities Java Main class (externally) from the command line using the SDK JAR. To do so, you would require an actual X9Utilities license, and your SDK license does not serve as a substitute for this license requirement. If executing X9Utilities from the command line is essential to your solution, we offer a substantial discount on X9Utilities to facilitate this need. This distinction emphasizes that while the SDK is a tool that can be used by Java developers to embed utility functions within their programs, X9Utilities remains as a standalone command-line product. This licensing difference is a key factor reflecting the intended use and separation of these two environments.

The X9Ware-SDK includes all Java classes that are utilized by our X9Utilities product. Hence X9Ware-SDK applications can directly invoke X9Utilities functions, as documented in the X9Utilities User Guide. These functions will perform exactly as they do for X9Utilities. The only significant difference is that the system exit is not performed on completion of the X9Utilities run. A system log will still be created for each uniquely invoked run.

Use of specific X9Utilities functions within your applications may simplify your development process and shorten your development cycle. However, when you leverage these capabilities, you will need to be aware of how these classes are organized, and will be responsible for the implementation testing necessary to ensure compatibility within your runtime environment.

X9Ware does not guarantee that the technical design of these classes will remain the same. There may be significant change or even complete redesigns over a period of time. You need to be aware that this possibility does exist and may result in future work on your part.

The high level code for X9Utilities is broken into several core classes:

- X9UtilMain contains the main() class and is responsible for initiation and termination and extends X9UtilBatch.
- X9UtilBatch implements common functions. All methods are public so they can be overridden as needed.
- X9UtilWorkUnit represents a single unit of work as performed within X9Utilities.
- Each worker function (write, export, validate, merge, compare, etc) is implemented as an independent class. These classes can be invoked directly or through X9UtilMain.

X9Ware has will provide current source code for X9UtilMain, X9UtilBatch, and X9UtilWorkUnit upon written request from licensed users. Source code for X9UtilMain (as of R5.03) is as follows:

```
package com.x9ware.utilities;

import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.x9ware.elements.X9Products;
import com.x9ware.error.X9Error;

/**
 * X9UtilMain is the static main class for X9Utilities, which is the command line interface for our
 * various batch products. The actual batch functions that are allowed will be determined based on
 * the current client license. Attempting to invoke a function that is not supported by the current
 * license will result in an abort. X9UtilMain is itself an extension of X9UtilBatch, where all
 * actually processing is performed. This design allows other SDK applications to extend X9UtilBatch
 * in a similar manner.
 *
 * @author X9Ware LLC. Copyright(c) 2012-2022 X9Ware LLC. All Rights Reserved. This is proprietary
 * software as developed and licensed by X9Ware LLC under the exclusive legal right of the
 * copyright holder. All licensees are provided the right to use the software only under
 * certain conditions, and are explicitly restricted from other specific uses including
 * modification, sharing, reuse, redistribution, or reverse engineering.
 */
public class X9UtilMain extends X9UtilBatch {
```

```
/*
 * Constants.
 */
public static final boolean ENVIRONMENT_OPEN_CLOSE_ENABLED = true;
public static final boolean ENVIRONMENT_OPEN_CLOSE_DISABLED = false;

/**
 * Logger instance.
 */
private static final Logger LOGGER = LoggerFactory.getLogger(X9UtilMain.class);

/**
 * X9UtilMain Constructor.
 */
public X9UtilMain() {
    /*
     * This constructor is always used when x9utilities is run from the command line. In this
     * launch scenario, we use the X9UTILITIES product name, which forces an x9utilities license
     * key to be located and applied to the runtime environment.
     */
    super(X9Products.X9UTILITIES, ENVIRONMENT_OPEN_ENABLED, ENVIRONMENT_CLOSE_ENABLED);
}

/**
 * X9UtilMain Constructor with explicitly defined environment open and close parameters.
 *
 * @param is_EnvironmentToBeOpenedAndClosed
 *         true or false
 */
public X9UtilMain(final boolean is_EnvironmentToBeOpenedAndClosed) {
    this(is_EnvironmentToBeOpenedAndClosed, is_EnvironmentToBeOpenedAndClosed);
}

/**
 * X9UtilMain Constructor with explicitly defined open and close parameters.
 *
 * @param is_EnvironmentToBeOpened
 *         true or false
 * @param is_EnvironmentToBeClosed
 *         true or false
 */
public X9UtilMain(final boolean is_EnvironmentToBeOpened,
                  final boolean is_EnvironmentToBeClosed) {
    /*
     * This constructor can only be invoked from an sdk application (it is never used from
     * x9utilities command line). This could be an sdk user application, but it could also be
     * x9assist running the utilities console. Either way, we now open the batch environment
     * with our sdk product name. We can logically do this since the invoking application has
     * already had its license key validated, hence it is appropriate to allow x9utilities to
     * launch without further license key validation. This is a core requirement, since an
     * x9assist user running the utilities console does not have an x9utilities license.
     */
    super(X9Products.X9SDK, is_EnvironmentToBeOpened, is_EnvironmentToBeClosed);
}

/**
 * Main as invoked directly from the command line. The only thing unique here is that we include
 * system exit which terminates the currently running JVM. Our launch method can otherwise be
 * used for more control over the runtime environment.
 *
 * @param args
 *         command line arguments
 */
public static void main(final String[] args) {
    /*
     * Run using try-with-resources to ensure we always close and system exit.
     */
    int exitStatus = EXIT_STATUS_ABORTED;
}
```

```
try (final X9UtilMain x9utilMain = new X9UtilMain()) {
    /*
     * Launch the x9utilities runtime.
     */
    final X9UtilWorkUnitList workUnitList = x9utilMain.launch(args);
    exitStatus = workUnitList.getExitStatus();

    /*
     * Generate list of all processing errors (writer, import, etc).
     */
    final List<X9Error> processingErrorList = workUnitList.getProcessingErrorList();
    if (processingErrorList != null && processingErrorList.size() > 0) {
        LOGGER.error("summary of processing errors:");
        for (final X9Error x9error : processingErrorList) {
            LOGGER.error(x9error.getFormulatedErrorString());
        }
    }
} catch (final Throwable t) {
    LOGGER.error("exception", t);
} finally {
    System.exit(exitStatus);
}
}
```


Using X9Objects

X9Object is an abstract class which defines that attributes of a single data record. An entire file can be loaded to an X9ObjectManager list, which provides transversal and management methods. If a file with 10,000 records is loaded to a list, then the list will have 10,000 x9objects.

X9Object has instantiated classes X9Object937 (for x9.37 records) and X9ObjectAch (for ach records). Within X9Object, there are several core fields that are exposed as public that are used extremely frequently. These are as follows:

x9ObjIdx	Contains the record index within the overall x9 file and is relative to one.
X9ObjType	Contains the record type. Instead of hard coding equal checks for certain record types, it is instead recommend that you use the constants that are provided in X9Ware-SDK class X9.
X9ObjFormat	Contains the record format which is a three character string and typically has a value of "000". The exception to this is those record types which have multiple logical formats, such as x9 credits and ach addendas.
X9ObjData	Contains the record data as a string which is stored in ASCII. This is typically 80 characters for x9 files and will always be 94 characters for ach files. Note that the type 52 image detail records for x9 files do not contain the actual images, but contain the x9 record data only.

The X9Object class has a large number of attributes which are exposed via getters and setters. You should closely review this functionality within the JavaDoc. There are a correspondingly long list of provided methods to logically access and manipulate the data that is stored within each x9object instance. X9Object provides methods to easily walk the records within the file in forward or backward directions.

X9ObjectManager is a companion class to X9Object provides a variety methods that are used to manage these record objects. In particular, X9ObjectManager allows x9objects to be efficiently stored in an array list which can then be used for a wide variety of purposes. Methods are provided to allow the first and last x9objects to be directly retrieved from the list. You can also retrieve any x9object by its corresponding index. Once you have access to an individual x9object, you can use it as the basis to then move forward and backward within the list.

X9ObjectManager is reusable, allowing a file file to be loaded to objects, then reset, and then reloaded from another file. You should closely review this functionality has well as a long list of methods that are provided to further access and manipulate x9object data.

An X9Object can be marked as deleted. This is done on a logical basis, which means that the instance is marked as deleted but physically remains within the X9ObjectManager object list. An x9object that is marked as deleted will be logically skipped over when walking objects in either a

forward or backward direction. An x9object that has been deleted can be reverted by changing the delete indicator.

- A common requirement is the need to reference a header record (file header, cash letter header, or bundle header) from a record within an item group. There are two ways to accomplish this:
- One alternative is to use the x9object “walk” methods which will traverse the x9objects in a reverse direction to obtain the needed header record. It should be noted that these methods will have associated overhead for larger files based on the repeated use of get previous.
- Another alternative is to use the x9object getter methods to directly obtain the associated file header, cash letter header, or bundle header. These methods utilize an internal X9Headers index that is stored within each x9object and is much more efficient than walking to the needed header records. Use of these x9object header getter methods requires that you first invoke the assign headers index method within X9ObjectManager prior to using indexing reference. Note this is always done by X9Validator since it is depending on those indexes being in place. If you are using X9ObjectManager lists but are not doing an x9 file validation, then you will need to assign the header indexes prior to using the getter methods.

Retrieving Fields within X9 Records

There are several ways to retrieve specific fields with x9 record types. You can choose from these alternatives based on your specific application requirements.

- 1) Create an x9 type specific object from an sdkObject or another source of x9 data. SdkObjects are typically created by X9SdkObjectFactory using X9SdkIO. Field values can then be retrieved directly from the x9 type specific object on a logical name basis.

```
X9Type01 t01 = new X9Type01(sdkBase, sdkObject.getX9AsciiRecord());
String destinationRouting = t01.immediateDestinationRoutingNumber;
String originationRouting = t01.immedateOriginRoutingNumber;
String createDate = t01.fileCreationDate;
String createTime = t01.fileCreationTime;
String fileIdModifier = t01.fileIdModifier;
```

- 2) Create an x9 type specific object from a previously created x9object and then retrieve the data fields on a logical name basis.

```
X9Type25 t25 = new X9Type25(x9o);
String routing = t25.payorBankRouting + t25.payorBankRoutingCheckDigit;
String micrOnUs = t25.micrOnUs;
String isn = t25.itemSequenceNumber;
BigDecimal amount = X9Decimal.getAsAmount(t25.amount);
```

- 3) Create an X9Item object which can then be used to retrieve a wide variety of commonly required x9 fields. X9Item fields can be populated in one of several manners. The easiest is to provide an x9object which contains the target item. The setItemFields method has other alternatives including the ability to parse a scanned MICR line to obtain individual fields. An example of setting fields from an x9object instance and then retrieving fields is as follows:

```
X9Item x9item = new X9Item();
x9item.setItemFields(x9o);
String micrAuxOnUs = x9item.getAuxOnus();
String micrRouting = x9item.getRouting();
String micrOnUs = x9item.getOnus();
```

- 4) Extract a specific field from an x9 record using the x9object and field number. This approach can be used to extract any field directly from the x9 byte array as a string and is most efficient when only one or two fields are to be referenced.

```
String bundleId = x9o.getFieldValue(sdkBase.r20BundleIdentifier);
```

- 5) Use the x9object to access any defined field within that x9 record type by field number. This approach has the benefit that it easily extracts the data from the x9 record itself (eg, from the 80 byte data record), and can thus be equally well for x9objects, sdkobjects, or x9 data that comes from any other input source.

```
/*
 * Get the x9field object for the bundle identifier.
 */
X9FieldManager x9fieldManager = sdkBase.getFieldManager();
X9Field x9field = x9fieldManager.getFieldObject(X9.BUNDLE_HEADER,
        X9.R20BundleIdentifier);

/*
 * Get the field value and field length.
 */
String bundleIdentifier = x9field.getStringValue(x9recordData).trim();
int fieldLength = x9field.getLength(x9recordData);
```

- 6) Sequentially walk all of the fields within an x9 record using the field walker.

```
/*
 * Walk all of the x9 records within the current x9 file.
 */
X9Object x9o = X9GuiAnchor.getFirstObject();
while (x9o != null) {
    /*
     * Walk all of the fields within the current x9 record.
     */
    final X9Field[] fieldArray = x9walk.getFieldArray(x9o);
    for (X9Field x9field : fieldArray) {
        /*
         * Get the field value.
         */
        String value = x9field.getFieldValueTrimmed(x9o);
    }
}
```

Modifying Fields within X9 Records

There are several ways to modify x9 record fields. These modification examples assume that you have loaded an x9 file to the heap using the facilities provided by X9ObjectManager. This is normally required since changing one x9 record can have impacts on other record types, so having access to all of the data is generally helpful. However, this is not an absolute requirement, and you can instead simply modify records during x9 read and write processing. For example, you can use an X9SdkObject to create an x9object that is not stored but only used for data read and write operations. You can choose from these alternatives based on your specific requirements.

- 1) Create an x9 type specific object from an sdkObject or another source of x9 data. SdkObjects are typically created by X9SdkObjectFactory using X9SdkIO. All field level modifications will be applied directly to the byte array used to allocate the x9 type specific object:

```
X9Type01 t01 = new X9Type01(sdkBase, sdkObject.getX9AsciiRecord());
t01.fileIdModifier = "B";
t01.modify();
```

- 2) Create an x9 type specific object from from an x9object and then modify fields. X9objects are typically created by X9Reader. All field level modifications are applied to the x9ObjData byte array within the supplied x9object.

```
X9Type25 t25 = new X9Type25(x9o);
t25.itemSequenceNumber = Long.toString(++itemSequenceNumber);
t25.modify();
```

- 3) Create an x9object and then directly modify individual fields directly within the x9object data byte array, on an offset and length basis using an X9SdkFid accessor. Commonly used accessor definitions can be found in X9FIDS, and you can create others as needed for your application. The rules class in these definitions are informational when using get/set and are directive when using the obtain/assign factory methods in X9SdkFid. X9Objects are typically created by X9Reader. All field level modifications are applied to the x9ObjData byte array within the referenced x9object.

```
private static final X9SdkFid R70_ITEM_COUNT = new X9SdkFid(X9SdkFid.RULES_ANY,
    X9.BUNDLE_TRAILER, 2, 4);
R70_ITEM_COUNT.setFieldValue(sdkBase, x9o.x9ObjData,
    X9Numeric.getAsString(value, 4));
```

- 4) Update all trailer records after a data value has been modified which impacts the counts and amounts that are present in the bundle trailers, cash letter trailers, and the file control trailer. You can optionally enable field level logging within X9TrailerManager as part of that constructor (the default is for that facility to be disabled). When field level logging is enabled, all updated fields (with before and after values) will be logged through X9ModifyManager, which can be subsequently used to retrieve all of the modifications that have been applied. Note that this code assumes that that x9 file has been loaded as x9objects to the heap. X9TrailerManager can also be

used to accumulate and optionally populate trailer record totals as individual x9 records are processed on a stream basis (see the next example).

```
X9TrailerManager x9trailerManager = new X9TrailerManager(sdkBase);
x9trailerManager.updateAllTrailerRecords();
```

5) X9TrailerManager also has methods that allow you to update the trailers when the x9 records are being directly read and written but have not be loaded to the heap.

```
/*
 * Accumulate and populate totals within the trailer records.
 */
x9trailerManager.accumulateAndPopulate(recordType, recordFormat,
                                       x9data);

/*
 * Build the 9 record from the current x9data and image.
 */
sdkObject.buildX9FromData(x9data);

/*
 * Write the current sdkObject to the x9 output file.
 */
currentByteCount += sdkIO.writeX9(sdkObject);
```

6) Replace an image and then update the image lengths in associated record types: The replacement image is stored within the x9object and will be subsequently used when this x9object is formatted and written to an output x9 file.

```
/*
 * Store the replacement image from a provided byte array.
 */
X9Type52Worker x9type52Worker = new X9Type52Worker(sdkBase);
x9o.setReplacementImage(tiffImage);

/*
 * Update the image length in the type 50 and 52 records.
 */
x9type52Worker.updateImageRecordLengths(x9o, tiffImage.length);
```

Credits And Trailer Totals

Credits (record type 61) and their impact on trailer records has been one of the more difficult topics associated with the creation of x9 files. These issues have been addressed with varying approaches, including the newer x9.100-187-2013 specification, where credit indicators are included in trailer records to actually define how and if credits are rolled into totals.

Earlier x9 specifications unfortunately are not nearly as clear cut and attempt to deal with the situation where credits are essentially extensions to those previously standards and not are incorporated into the base definition and core design.

For many x9 variant specifications, there is really no reason to include credits in trailer totals whatsoever. The typical image cash letter consists of a credit offset by debits. In this situation, including the credit amount in the totals records only serves to unnecessarily double the amount fields. Similarly, there is no absolute need to include the credit in the item counts either, since the credit record will be accounted for in the total record count in the type 99 file control trailer, so its presence is tracked by the overall file counters. The basis of this (hopefully logical) position is that items are defined as record types 25 (forward present) and 26 (returns) and comprise the trailer record item count totals, while credit record type 61 would be excluded from those totals.

Unfortunately, for some x9 variant definitions, this is not the case.

The X9Ware-SDK has two classes that manage totals. The first is X9TrailerTotals which is responsible for accumulating an individual set of running totals, where debits and credit counts and amounts are tracked separately. The second is X9TrailerManager which is responsible for accumulating, validating, and populating the various totals in the x9 trailer records. X9TrailerManager keeps separate running totals for each of the levels within an x9 file (batch, cash letter, and overall file).

X9TrailerManager includes appropriate logic to support for several standards where it is very clear how credits add into trailer record counts and amounts. This includes the x9.100-187-2013 specification, where a credit indicator totals determines how totals are accumulated and populated, and x9.100-180, where separate debit and credit counts are physically populated.

However, there are times when this standard X9TrailerManager logic is not sufficient to meet specific needs. This typically happens when creating an image cash letter that includes credits for an x9 variant that adds type 61 credit counts and/or amounts into the various trailer records.

X9TrailerManager allows you to provide direction as the impact that credits will have on the bundle, cash letter, and file control trailer records. These actions can be specified as a group or can be applied individually to each of the trailer record types. This allows X9TrailerManager to support even the worst case scenarios, for example when credits are added into the bundle totals but are not added into the higher level cash letter or file control item totals. Or similarly when

credits are added into the item count totals at the bundle, cash letter, and file control levels, but are then not included in the item amount totals. X9TrailerManager can support these (and other) variations by setting action directives at the appropriate levels.

In order to take control of trailer totals, you must first access the X9TrailerManager instance that is being used to accumulate and populate your totals.

If your x9 file is being created from directly through X9SdkIO, this is done as follows:

```
final X9TrailerManager x9trailerManager = sdkIO.getTrailerManager();
```

If your x9 file is being created through X9Writer, this can be done as follows:

```
final X9TrailerManager x9trailerManager = x9writer.getTrailerManager();
```

Once you have a reference to the trailer manager, you can now set actions to explicitly identify how you want your trailer totals accumulated and then populated. The following action codes are defined within X9TrailerManager:

```
public static final int CREDITS_DEFAULT_ACTION = 0;
public static final int CREDITS_ADD_TO_COUNT = 1;
public static final int CREDITS_ADD_TO_AMOUNT = 2;
public static final int CREDITS_ADD_TO_COUNT_AND_AMOUNT = 3;
```

X9TrailerManager then provides the following getters and setters to allow you to provide the direction needed for your specific requirements:

- get/set CreditsBundleTrailerAction
- get/set CreditsCashLetterTrailerAction
- get/set CreditsFileControlTrailerAction

Using X9Writer

X9Writer is a high level I/O interface which can be used to create an x9 file on an item by item basis. It includes methods to open the output x9 file, add an item with optional addendums, add type 68 user records of your design, and to then close the output x9 file. X9Writer can be used to create ICL (forward presentment) files or ICLR (return item) files.

The power of X9Writer comes from its ability to control the format and header record values from either internal or external sources. This design allows an x9 file to be created per the requirements of the receiving financial institution without having to hard-wire those definitions within your application program. X9Writer allows these parameters to be assigned from an external XML file which allows all x9 attributes to be easily defined and manipulated. HeaderXml control fields various field level formatting and populates values for the file header, cash letter header, and bundle header records. This facility contained a large number of attributes which control the overall creation of the output x9 file. HeaderXml values can also be overridden using setter methods. See the appendix for a full definition of available HeaderXml file content.

You can optionally include addendums using this interface, should you have that requirement. Individual addendums are created as an array of fields which are added to the item prior to being written. By attaching the addendums to the item, the addendum count can be queried by X9Writer and included in the type 25 or type 31 record.

Items can be presented in one of several ways. First you can provide a CSV array of the fields that explicitly define the new item (eg, an array of fields for a type 25 record). This approach gives you very detailed control over the x9 item record that is created. A second approach is to populate an X9Item object and provide that to X9Writer. This can be an easier approach and has the advantage that X9Item can be directly populated from either an x9object or can be set from your x9 data including MICR line parse.

X9Writer controls item level batching and totals are automatically accumulated and used to populate the appropriate records and fields in the x9 trailer records.

Refer to the X9DemoWriter example program which further highlights the use of X9Writer.

X9Ware-SDK Code Examples

Our X9Ware-SDK examples are designed to show common use cases, but cannot cover the myriad of things that can be performed by our X9Ware-SDK. You can review the X9Ware-SDK API to get a feeling for the large number of classes and methods that are available. It would be impossible to provide code examples for all of the operations and variations that can be performed by the X9Ware-SDK. Our X9Assist application is a real world example of the types of things that can be done by the X9Ware-SDK, since it is built directly on top of the X9Ware-SDK and is an excellent illustration of what can be done. To this end, we have published the actual source code to our X9Utilities product, which is similarly built on top of the X9Ware-SDK. Please take a close look at both our SDK examples and the X9Utilities source code, since together they provide good insight into the overall capabilities. However, if you need to solve a specific issue that is not represented by these examples, please contact us and let us know exactly what you need. In those cases, we may provide you a code snippet that addresses your specific requirement, expand one of our existing X9Ware-SDK examples to demonstrate your need, or perhaps even develop and publish a new X9Ware-SDK example program that fully satisfies the requested capability. Our goal is to ensure that there are adequate examples to allow the X9Ware-SDK to be fully reviewed, and we believe that these actions will be helpful for both prospective and current customers.

The X9Ware website and the X9Ware-SDK distribution packages include source code for various X9Ware-SDK examples. You should review these since they provide examples of coding solutions for common user functions. We would be glad to extend these examples to address your specific questions. There are two source code samples:

- The first is a zip package of our X9Ware-SDK example programs. These samples are working programs and can be reviewed in combination with the X9Ware-SDK JavaDoc that is available as part of our X9Ware-SDK Documentation. We strive to make our examples package as complete as possible. Please provide comments on possible additions and improvements.
- The second is a zip package of our X9Utilities source code. X9Ware has made this source code public knowing that a complete X9Ware-SDK based application will be helpful. This source represents the actual production version of our X9Utilities product. These are not just theoretical examples, but the actual working source code for X9Utilities that will be updated with each release as they are published. We believe that this source may well be the best example of the X9Ware-SDK, since our design goal for X9Utilities is to reference and leverage functionality that is implemented within the X9Ware-SDK. Hence the technical approach has been to minimize code within X9Utilities by pushing as much of the implementation as possible into the core X9Ware-SDK. You will see this in the X9Utilities source code, where the design thus allows those core X9Ware-SDK classes to be incorporated in your applications.

Number	Java Example	Description
1	X9DemoWriter	X9DemoWriter is an example of our x9writer technology, which is the easiest way to create to create an x9.37 file. This specific example reads an input CSV file to obtain the items

Number	Java Example	Description
		<p>which will be written as x9. X9DemoWriter achieves this high level of simplification by defining the attributes of the x9 file to be created in what we call the headerXml file. In this way, the application program only needs to be concerned with the logical items to be written, making x9writer responsible for all other details. The headerXml file identifies the x9.37 specification to be created, along with other complex topics such as addenda creation, image attachment, and optional credit insertion. This design becomes especially advantageous when creating files for multiple financial institutions, since it allows your application program to work on a logical item basis. This means that you application program can build x9.37 files for any financial institution, regardless of format, with absolutely no changes to your program. HeaderXml has 100+ parameters that define the contents of the x9 header records (file header, cash letter header, and bundle header),bundling of items, and the many unusual requirements that must be satisfied. X9Writer is responsible for the actual creation of all record types.</p>
2	X9ConstructX9	<p>X9ConstructX9 is an alternative to X9DemoWriter. The output x9 file is written on a record by record basis, with (almost) all records formatted explicitly within the program itself. Although this provides complete control over each individual record that is written, this approach is much more complex. It is our recommendation that you should instead work at the logical as demonstrated by X9DemoWriter. The X9ConstructX9 still takes advantage of several core components of our X9Ware-SDK. First is that it uses the x9factory to populate fields based on the selected x9 specification (for example x9.37 DSTU, x9.100-187-2008, etc). This means that you can provide values for logical fields for all fields that exist in a certain record type, even when the specification that you have selected may only require a subset of those fields. Second is that it allows the X9Ware-SDK to calculate and populate content of all trailer records (batch trailer, cash letter trailer, and file trailer) subject to data content and the x9 specification that is being used.</p>
3	X9ConstructAch	<p>X9ConstructAch is an example of our achWriter technology, which allows ACH files to be created at a very high level (similar to our approach of using x9writer for creation of x9.37 files). X9ConstructAch specifically reads an x9.37 file to get logical items that are then written as an ACH file. Use</p>

Number	Java Example	Description
		<p>of achWriter allows the output file to be written as any desired Standard Entry Class (SEC). AchWriter also supports the attachment of needed addenda records, subject to the current entry class. AchWriter provided automatic batching and will reorder input as needed to allow transactions to be grouped within batches by entry class. AchWriter also computes hash totals and appends nines records as required to pad the output file as required by ACH standards.</p>
4	X9VerifyX9	<p>X9VerifyX9 is an extensive example of how to perform a variety of functions against an existing x9.37 file. The file is first loaded into x9objects that are stored using our object manager. It then runs our validation process against the file, in the same manner that would be done by our X9Assist desktop program. Any identified errors are written to the system log. Totals are accumulated for all debits and credits within the file and are summarized to the log. Various record types are listed to the log. Several records are both modified and deleted. The result is then written to a new output file, with the trailer records recalculated based on the modifications that have been made.</p>
5	X9VerifyAch	<p>X9VerifyAch is an extensive example of how to perform a variety of functions against an existing ach file. The file is first loaded into x9objects that are stored using our object manager. It then runs our validation process against the file, in the same manner that would be done by our X9Assist desktop program. Any identified errors are written to the system log. Totals are accumulated for all debits and credits within the file and are summarized to the log. Various record types are listed to the log. Several records are both modified and deleted. The result is then written to a new output file, with the trailer records recalculated based on the modifications that have been made.</p>
6	X9ReformatX9	<p>X9ReformatX9 is an example of reading an input x9.37 file to get logical items which are then written using x9writer. This is a very powerful example because it allows the items to be extracted and then rewritten by the application program. By using x9writer, the output x9.37 file can be written in a different x9 specification than the input, and it would similarly allow an offsetting credit to be either removed or added. Finally, it would be possible to modify the items in some manner (if needed) between input and output. This approach may be application to many use case scenarios.</p>

Number	Java Example	Description
7	X9ModifyX9	X9ModifyX9 is an example of reading an x9 file, applying modifications, and then writing that output to another external x9 file. The example is based on an input file which is modified and written directly to the output file.
8	X9ReadX9AsItems	X9ReadX9AsItems is a static class which reads an input x9.37 file and loads the items into an array list. This is common functionality utilized by several other X9Ware-SDK example programs.
9	X9DrawImage	X9DrawImage is a demo of creating front and back images dynamically using our drawing tools. Images are created using a template which you can define based on your requirements using a tool of your choosing such as GIMP, Paint, or Photoshop. Templates are stored in our template library in a common image format such as PNG. The X9Ware-SDK drawing tools allow you to add various fields to each created image such as name/address, payee, memo, signature, date, check number, MICR line, and so forth. Once created, the images can be converted to standard x9.100-181 exchange format.
10	X9ReadCsvWriteX9	X9ReadCsvWriteX9 will read a csv file with corresponding images for each item stored in an associated image folder. The data and images are then used to create x9 records which are written to an x9 output file. X9Ware-SDK methods are used which would allow the csv input records to be examined or modified as the x9 file is created and written.
11	X9ReadX9ByRecordType	X9ReadX9ByRecordType will read an x9 file and use type objects to map and retrieve the specific fields as defined at the x9 record type level. Note that the field level classes also support modify, so they can be used to modify individual fields and create a modified x9 file.
12	X9ReadX9WriteCsv	X9ReadX9ByRecordType will read an x9 file and use type objects to map and retrieve the specific fields as defined at the x9 record type level. Note that the field level classes also support modify, so they can be used to modify individual fields and create a modified x9 file.
13	X9ReadX9WriteX9	X9ReadX9WriteX9 will read an x9 file and create an output x9 file with associated images. This is a good example of reading an x9 file which is then loaded to resident x9objects. All fields within each record are examined using walk. This example could be easily extended to allow individual fields to be modified as needed. The possibly modified x9 file is written from the resident x9objects array.

Number	Java Example	Description
14	X9DemoWriterThreaded	X9DemoWriterThreaded provides a sample of running x9writer in a multi-threaded environment, where twenty (20) files are created in parallel. It provides insight into how the X9Ware-SDK can be run across concurrent threads, and also shows the power of the X9Ware-SDK to run in a high volume production environment. As a demonstration of X9Ware-SDK, this sample program runs in less then 30 seconds when creating twenty files with 5,000 items each, on a Dell laptop with the images stored on a local SSD.
15	X9MakeX9	X9MakeX9 is an example of the make process which reads a use case file (which must be in CSV format) and creates an output x9 file. An xml reformatter is loaded which defines the make parameters. This xml file can be created, tested, and maintained using X9Assist.
16	X9GenerateX9	X9GenerateX9 is an example of the generate process which reads a CSV file that contains generate specific columns and creates an output x9 file. The column requirements may vary from release to release and are logged as information. An xml generator is loaded which defines the generate parameters. This xml file can be created, tested, and maintained using X9Assist.
17	X9ScrubX9	X9ScrubX9 is an example of the scrub process which reads an x9 file and creates a scrubbed x9 output file. An xml scrub configuration is loaded which defines the scrub parameters to be applied. This xml file can be created, tested, and maintained using X9Assist.
18	X9PrintX9	X9PrintX9 is an example of the image print process which reads an x9 file and creates an image print stream which is routed to a selected printer. The example includes either interactive print (where a GUI dialog is invoked to select the printer) or silent print (where output is written to a specific printer). An xml print configuration is loaded which defines the print parameters. This xml file can be created, tested, and maintained using X9Assist.
19	X9PaidEndorsement	X9PaidEndorsement is an example of adding a paid endorsement stamp to a back side image by using X9DrawTools. The paid endorsement stamp is a series of text lines that are drawn on the image rotated 90 and centered within the image. Placement is based on a right side margin that is specified in inches. The paid endorsement stamp can be a variable number of lines with each having a define font, style, and size.

Number	Java Example	Description
20	X9Utilities	<p>X9Ware has made this source code public knowing that a complete X9Ware-SDK based application will be helpful. This source represents the actual production version of our X9Utilities product. These are not just theoretical examples, but the actual working source code for X9Utilities that will be updated with each release as they are published. We believe that this source may well be the best example of our X9Ware-SDK, since our design goal for X9Utilities is to reference and leverage functionality that is implemented within the X9Ware-SDK.</p>

Rules Overview

X9Ware LLC has developed a very powerful and what we believe to be a very unique rules engine that can be used to document and then validate the format of X9.37, ACH, and CPA005 files. The format of these rule definitions are proprietary to X9Ware and are the result of substantial ongoing design and improvement. This documentation is confidential to X9Ware and can only be shared within an organization that has established a non-disclosure agreement (NDA) with X9Ware. Please do not share this information outside of your organization. Any other use, including reverse engineering into other formats for other purposes, is expressly prohibited.

This documentation is offered by X9Ware to provide insight into the usage of our rules based technologies. Customers can use this information to leverage the X9Ware rules engine to validate your internal x9 variants. As part of Extended Support, X9Ware provides the needed assistance to allow your organization to implement validation of your x9 variants using our rules engine.

Rules are loaded and evaluated dynamically by the X9Ware-SDK. There is no need to run any utilities to evaluate and populate the rules. Descriptive error messages will be issued by the X9Ware-SDK if there are errors within a rules specification.

The mention of “x9” below applies equally to all file dialects that are supported with the SDK (X9.37, ACH, and CPA005). The X9 rules engine implements the following proprietary design principles:

- An x9 specification can be either a basis or an extension document.
- An x9 basis defines a core x9 specification.
- An x9 extension defines a new x9 specification that is built upon a defined base. Only the differences need to be defined. This approach substantially reduces ongoing maintenance, since a change basis is automatically applied to any defined extensions. It also allows you to quickly and easily understand what is different in the extension specifications.
- Each x9 specification has a set of x9 controls which define the high level attributes associated with the x9 rules. For example, the x9 controls might indicate if the allowable character set is “EbcDic”, “Ascii”, or “either”.
- An x9 base specification must contain an x9controls definition and need only define those values that vary from the system defaults.
- An x9 extension specification must contain an x9controls definition and need only define those values that vary from the base specification.
- The x9 specification is defined as consisting of a series of x9 record types.
- Each x9 record is defined as consisting of a series of x9 fields.
- Each x9field is defined with its specific attributes and validation criteria.

The TIFF rules engine similarly implements the following proprietary design principles:

- Each TIFF specification has a set of TIFF controls which define the high level attributes associated with the TIFF rules. For example, the TIFF controls might indicate that duplicate tags are accepted or are not accepted.
- Each TIFF specification can have separate set of TIFF rules for Black White versus Gray Scale images.
- Each set of TIFF rules contains a list of TIFF edits that are applied to tags when they are present within an image.
- Each set of TIFF rules contains a list of TIFF tags that are mandatory for each image.
- The TIFF specification contains a list of descriptions of all possible tags that can be present across all images within the x9 file.

X9 Configurations

An X9 configuration defines the components that, when taken as a group, define the validations that will be applied to the x9 file. A list of standard configurations are defined internally within the X9Ware-SDK and can be defined externally as config.xml within the “xml” folder.

Every configuration consists of the following elements:

Configuration name	Your assigned configuration name.
X9 rules file	Contains a list of the rules that are applied to the records and fields within the currently loaded file. These rules are used to identify and format the associated error messages.
Tiffrules file	Contains a list of the rules that are applied to the tiff images within the x9 file. These rules are used to generate image related errors.
Messages file	Contains a list of the error messages and their associated severity levels. These rules are used to assign the severity level of the errors that are generated during the validation process. Each error has an assigned severity level of Error, Warn, Info, or OK.
Entry Type	Defines the configuration as either “System” or “User”. System entries are automatically defined within the X9Assist environment and cannot be modified by the user. User entries can be created using the Configuration Editor and will be automatically retained across user sessions and X9Assist release installations.

Custom configuration entries can be defined within the external config.xml definition or can be directly populated at run time into the configuration map.

X9ConfigManager defines the standard (system) configuration map as follows:

```

addMapEntry(X9.X9_100_187_2008_CONFIG,
            "x9rules_x9.100-187.xml",
            TIFFRULES_100_187_2014, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_100_187_UCD_2008_CONFIG,
            "x9rules_x9.100-187_UCD.xml",
            TIFFRULES_100_187_2014, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_100_187_2013_CONFIG,
            "x9rules_x9.100-187-2013.xml",
            TIFFRULES_100_187_2014, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_100_187_UCD_2013_CONFIG,
            "x9rules_x9.100-187_UCD-2013.xml",
            TIFFRULES_100_187_2014, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_37_CONFIG,
            "x9rules_x937.xml",
            TIFFRULES, MESSAGES, X9C.SYSTEM);

```

```

addMapEntry(X9.X9_CPA_015_CONFIG,
            "x9rules_x9.100-187_CCD.xml",
            TIFFRULES_100_187_2014, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_EC_ACH_CONFIG,
            "x9rules_eastCaribbeanAch.xml",
            TIFFRULES, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_EEX_CONFIG,
            "x9rules_eex.xml",
            TIFFRULES, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_FRB_CONFIG,
            "x9rules_frb.xml",
            TIFFRULES_FRB, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_SVPCO_CONFIG,
            "x9rules_svpco.xml",
            TIFFRULES_100_187_2014, MESSAGES, X9C.SYSTEM);
addMapEntry(X9.X9_VIEWPOINTE_CONFIG,
            "x9rules_viewpointe.xml",
            TIFFRULES_100_187_2014, MESSAGES, X9C.SYSTEM);

```

The configuration is stored within a TreeMap. Custom entries can be dynamically added to this map using `addMapEntry`. X9Ware-SDK implementations need to consider their approach for custom entries as being either externally loaded or internally supplied. Some considerations are as follows:

- X9ConfigLoader should only be provided the base name (with the extension) for the x9rules, tiff rules, and messages. It does not accept a fully qualified path. This is because the Configuration Loader utilizes a multi-step load process that first looks at embedded resources within the X9Ware-SDK JAR, and then defers to external resources. By going to embedded resources first, the X9Ware-SDK can be fully self defining and not dependent on the file system.
- When rules are not found as embedded JAR resources, the Configuration Loader then uses the provided base name to construct a file reference within the program launch folder. That reference will become / program launch folder / rules / x9rules / rules-file-name.xml. The rules can alternatively be stored in the application home folder, which would be / home folder / rules / x9rules / rules-file-name.xml.
- You can determine the launch and home folder names, because they are logged by the X9Ware-X9Ware-SDK during startup.
- Your new configuration name should be added after your X9Ware-SDK application has loaded the standard xml configuration files (`X9SdkRoot.loadXmlConfigurationFiles()`) but before the bind is issued.
- Your new configuration should also be added as SYSTEM and not USER. This is because your configuration would be user defined and not part of our internally defined rule sets.
- Your bind must reference the newly defined configuration name.
- If you store the rules in the file system, we highly suggest that you create your own qualified reference to the rules themselves, and do an “`isFile()`” to ensure that the rules exist as expected. You would abort if the rules are not found. You can also use our utility method `X9FileUtils.existsWithPathTracing()` to check if the rules exist, which provides some additional tracing when the file location is not found.

- Finally, an alternative is to use the Java standard utility “jar -uf” to add your rule definitions to the X9Ware-SDK JAR itself, which makes your rules self defining, thus eliminating the needed to store the rules within the file system.

X9 Rules

An X9 base specification contains the following XML definitions:

Xml Definition	Usage	Notes
x9Controls	Defines the x9 controls (attributes) that are to be applied to this x9 specification.	Each control value will be assigned its default value when otherwise omitted.
records	Defines a list of x9 records that comprise this x9 specification.	Required.
record	Defines an individual x9 record and its associated attributes and fields.	Required.
field	Defined an individual x9 field and its associated attributes.	Required.

An X9 extension specification contains the following XML definitions:

Xml Definition	Usage	Notes
x9Controls	Defines the x9 controls (attributes) that are to be applied to this x9 specification.	Each control value will default to the value defined at the basis level when otherwise omitted.
basis	Defines the base document against which this specification is being applied. The presence of the the basis XML definition is used to identify a base versus extension x9 specification.	Required.
overrides	Defines a list of x9 records that are applied as overrides against the base specification.	Required.
record	Defines an individual x9 record and its associated attributes and fields. Only the fields that are different (when compared to the base) must be defined.	Required.

field	Defined an individual x9 field and its associated attributes.	Required.
-------	---	-----------

X9 Rules – Base Specification Example

The following is an example of an X9 base specification. Note that all controls, record types, and fields are defined.

```
<?xml version="1.0" encoding="UTF-8"?>
<x9rules>

  <x9Controls>
    <x9Specification>DSTU 2003</x9Specification>
    <characterSet>either</characterSet>
    <maximumFileSize>0</maximumFileSize>
    <fieldZeroPresence>optional</fieldZeroPresence>
    <fieldZeroFormat>bigEndian</fieldZeroFormat>
      <dateMinimumYear>1993</dateMinimumYear>
      <dateMaximumYear>2099</dateMaximumYear>
      <dateWindowMinusDays>1095</dateWindowMinusDays>
      <dateWindowPlusDays>95</dateWindowPlusDays>
    <userFieldsValidated>true</userFieldsValidated>
    <reservedFieldsValidated>true</reservedFieldsValidated>
    <multipleLogicalFilesAllowed>false</multipleLogicalFilesAllowed>
    <iclCollectionTypes>=00|01|02|12</iclCollectionTypes>
    <iclRecordTypeIndicators>=E|I|F</iclRecordTypeIndicators>
    <iclrCollectionTypes>=03|04|05|06</iclrCollectionTypes>
    <iclrRecordTypeIndicators>=E|I|F</iclrRecordTypeIndicators>
  </x9Controls>

  <records>

    <x9record>
      <type>00</type>
      <format>0</format>
      <name>Invalid Record Type</name>
      <length>f80</length>
      <field> <item>x00.01-p001-l002-mandatory-notModifiable</item>
        <edit>n</edit>
        <name>Record Type</name> </field>
      <field> <item>x00.02-p003-l078-mandatory-modifiable</item>
        <edit>none</edit>
        <name>Reserved</name> </field>
    </x9record>
```

```

<x9record>
  <type>01</type>
  <format>0</format>
  <name>File Header Record</name>
  <length>f80</length>
  <field> <item>x01.01-p001-l002-mandatory-notModifiable</item>
    <edit>n</edit>
    <name>Record Type</name> </field>
  <field> <item>x01.02-p003-l002-mandatory-modifiable</item>
    <edit>n</edit>
    <values>=1|2|3</values>
    <name>Standard Level</name> </field>
  <field> <item>x01.03-p005-l001-mandatory-modifiable</item>
    <edit>a</edit>
    <values>=P|T</values>
    <name>Test File Indicator</name> </field>
  <field> <item>x01.04-p006-l009-mandatory-modifiable</item>
    <edit>n</edit>
    <name>Immediate Destination Routing Number</name> </field>
  <field> <item>x01.05-p015-l009-mandatory-modifiable</item>
    <edit>n</edit>
    <name>Immediate Origin Routing Number</name> </field>
  <field> <item>x01.06-p024-l008-mandatory-modifiable</item>
    <edit>yyyymmdd</edit>
    <name>File Creation Date</name> </field>
  <field> <item>x01.07-p032-l004-mandatory-modifiable</item>
    <edit>Timehhmm</edit>
    <name>File Creation Time</name> </field>
  <field> <item>x01.08-p036-l001-mandatory-modifiable</item>
    <edit>a</edit>
    <values>=Y|N</values>
    <name>Resend Indicator</name> </field>
  <field> <item>x01.09-p037-l018-conditional-modifiable</item>
    <edit>ans</edit>
    <name>Immediate Destination Name</name> </field>
  <field> <item>x01.10-p055-l018-conditional-modifiable</item>
    <edit>ans</edit>
    <name>Immediate Origin Name</name> </field>
  <field> <item>x01.11-p073-l001-conditional-modifiable</item>
    <edit>an</edit>
    <name>File ID Modifier</name> </field>
  <field> <item>x01.12-p074-l002-conditional-modifiable</item>
    <edit>a</edit>
    <name>Country Code</name> </field>
  <field> <item>x01.13-p076-l004-conditional-modifiable</item>
    <edit>ans</edit>
    <name>User Field</name> </field>
  <field> <item>x01.14-p080-l001-mandatory-modifiable</item>

```

```

        <edit>b</edit>
      <name>Reserved</name> </field>
</x9record>

....
....
....
....

<x9record>
  <type>99</type>
  <format>0</format>
  <name>File Control Record</name>
  <length>f80</length>
  <field> <item>x99.01-p001-l002-mandatory-notModifiable</item>
    <edit>n</edit>
    <name>Record Type</name> </field>
  <field> <item>x99.02-p003-l006-mandatory-modifiable</item>
    <edit>n</edit>
    <edit>CashLetterCount</edit>
    <name>Cash Letter Count</name> </field>
  <field> <item>x99.03-p009-l008-mandatory-modifiable</item>
    <edit>n</edit>
    <edit>TotalRecordCount</edit>
    <name>Total Record Count</name> </field>
  <field> <item>x99.04-p017-l008-mandatory-modifiable</item>
    <edit>n</edit>
    <edit>TotalItemCount/0</edit>
    <name>Total Item Count</name> </field>
  <field> <item>x99.05-p025-l016-mandatory-modifiable</item>
    <edit>n</edit>
    <edit>TotalFileAmount</edit>
    <name>Total File Amount</name> </field>
  <field> <item>x99.06-p041-l014-conditional-modifiable</item>
    <edit>ans</edit>
    <name>Immediate Origin Contact Name</name> </field>
  <field> <item>x99.07-p055-l010-conditional-modifiable</item>
    <edit>n</edit>
    <name>Immediate Origin Contact Phone Number</name> </field>
  <field> <item>x99.08-p065-l016-mandatory-modifiable</item>
    <edit>b</edit>
    <name>Reserved</name> </field>
</x9record>

</records>

</x9rules>

```


X9 Rules – Extension Specification Example

```

<?xml version="1.0" encoding="UTF-8"?>
<x9rules>

  <x9Controls>
    <x9Specification>FRB 2003</x9Specification>
    <characterSet>EbcDic</characterSet>
    <maximumFileSize>2048</maximumFileSize>
    <fieldZeroPresence>required</fieldZeroPresence>
    <userFieldsValidated>false</userFieldsValidated>
    <reservedFieldsValidated>false</reservedFieldsValidated>
  </x9Controls>

  <basis>
    <base>x9rules_dstu_2003.xml</base>
  </basis>

  <overrides>

    <x9record>
      <type>01</type>
      <format>0</format>
      <name>File Header Record</name>
      <length>f80</length>
      <field> <item>x01.01-p001-l002-mandatory-notModifiable</item>
        <edit>n</edit>
        <name>Record Type</name> </field>
      <field> <item>x01.02-p003-l002-mandatory-modifiable</item>
        <edit>n</edit>
        <values>=3</values>
        <name>Standard Level</name> </field>
      <field> <item>x01.08-p036-l001-mandatory-modifiable</item>
        <edit>a</edit>
        <values>=N</values>
        <name>Resend Indicator</name> </field>
    </x9record>

    <x9record>
      <type>10</type>
      <format>0</format>
      <name>Cash Letter Header Record</name>
      <length>f80</length>
      <field> <item>x10.02-p003-l002-mandatory-modifiable</item>
        <edit>n</edit>
        <values>=1|2|3</values>
        <name>Collection Type Indicator</name> </field>
      <field> <item>x10.08-p043-l001-mandatory-modifiable</item>

```

```

        <edit>a</edit>
        <values>=I</values>
        <name>Cash Letter Record Type Indicator</name> </field>
    <field> <item>x10.09-p044-1001-conditional-modifiable</item>
        <edit>an</edit>
        <values>=G</values>
        <name>Cash Letter Documentation Type Indicator</name> </field>
    <field> <item>x10.10-p045-1008-conditional-modifiable</item>
        <edit>an</edit>
        <edit>CashLetterIdIsUnique</edit>
        <name>Cash Letter ID</name> </field>
    <field> <item>x10.13-p077-1001-conditional-modifiable</item>
        <edit>an</edit>
        <values>=A|B|C|D|H|I|J|K|L|M|N|R|S|X|1|3|7|8</values>
        <name>Fed Work Type</name> </field>
</x9record>

<x9record>
    <type>20</type>
    <format>0</format>
    <name>Bundle Header Record</name>
    <length>f80</length>
    <field> <item>x20.02-p003-1002-mandatory-modifiable</item>
        <edit>n</edit>
        <values>=1|3</values>
        <name>Collection Type Indicator</name> </field>
    <field> <item>x20.10-p055-1009-conditional-modifiable</item>
        <edit>n</edit>
        <validate>>false</validate>
        <name>Return Location Routing Number</name> </field>
</x9record>

<x9record>
    <type>25</type>
    <format>0</format>
    <name>Check Detail Record</name>
    <length>f80</length>
    <field> <item>x25.09-p073-1001-conditional-modifiable</item>
        <edit>an</edit>
        <values>=G</values>
        <name>Documentation Type Indicator</name> </field>
</x9record>

<x9record>
    <type>28</type>
    <format>0</format>
    <name>Check Detail Addendum C Record</name>
    <length>f80</length>

```

```
<field> <item>x28.09-p040-l001-conditional-modifiable</item>
      <edit>an</edit>

<values>=A|B|C|D|E|F|G|H|I|J|K|L|M|N|P|Q|R|S|T|W|X</values>
      <name>Return Reason</name> </field>
</x9record>

<x9record>
  <type>31</type>
  <format>0</format>
  <name>Return Record</name>
  <length>f80</length>
  <field> <item>x31.06-p042-l001-conditional-modifiable</item>
        <edit>an</edit>

  <values>=A|B|C|D|E|F|G|H|I|J|K|L|M|N|P|Q|R|S|T|W|X</values>
        <name>Return Reason</name> </field>
  <field> <item>x31.08-p045-l001-conditional-modifiable</item>
        <edit>an</edit>
        <validate>>false</validate>
        <name>Return Documentation Type Indicator</name> </field>
</x9record>

  <x9record>
  <type>35</type>
  <format>0</format>
  <name>Return Addendum D Record</name>
  <length>f80</length>
  <field> <item>x35.09-p040-l001-conditional-modifiable</item>
        <edit>an</edit>
        <edit>CompareLastAddendumReturnReasonToItem</edit>

  <values>=A|B|C|D|E|F|G|H|I|J|K|L|M|N|P|Q|R|S|T|W|X</values>
        <name>Return Reason</name> </field>
</x9record>

<x9record>
  <type>40</type>
  <format>0</format>
  <name>Account Totals Detail Record</name>
  <length>f80</length>
  <allowed>>false</allowed>
</x9record>

<x9record>
  <type>41</type>
  <format>0</format>
  <name>Non-Hit Totals Detail Record</name>
```

```

    <length>f80</length>
    <allowed>false</allowed>
  </x9record>

  <x9record>
    <type>50</type>
    <format>0</format>
    <name>Image View Detail Record</name>
    <length>f80</length>
    <field> <item>x50.02-p003-l001-mandatory-modifiable</item>
      <edit>n</edit>
      <values>=0|1</values>
      <name>Image Indicator</name> </field>
    <field> <item>x50.05-p021-l002-mandatory-modifiable</item>
      <edit>nb</edit>
      <values>=0</values>
      <name>Image View Format Indicator</name> </field>
    <field> <item>x50.06-p023-l002-mandatory-modifiable</item>
      <edit>nb</edit>
      <values>=00</values>
      <name>Image Compression Algorithm Indicator</name> </field>

    <field> <item>x50.09-p033-l002-mandatory-modifiable</item>
      <edit>n</edit>
      <values>=0</values>
      <name>View Descriptor</name> </field>
    <field> <item>x50.10-p035-l001-mandatory-modifiable</item>
      <edit>nb</edit>
      <values>=0</values>
      <name>Digital Signature Indicator</name> </field>
  </x9record>

  <x9record>
    <type>52</type>
    <format>0</format>
    <name>Image View Data Record</name>
    <length>v117</length>
    <field> <item>x52.09-p085-l001-mandatory-modifiable</item>
      <edit>nb</edit>
      <values>=0</values>
      <name>Clipping Origin</name> </field>
    <field> <item>x52.18-p000-l007-mandatory-notModifiable</item>
      <edit>nb</edit>
      <edit>MinimumImageLength/250</edit>
      <edit>MaximumImageLength/200000</edit>
      <edit>MaximumCombinedImageLength/400000</edit>
      <edit>CompareToImageDetailImageLength</edit>
      <edit>CompareToImageDetailImageIndicator</edit>
  </x9record>

```

```
        <edit>ImagePresenceBasedOnDocType</edit>
        <variableLengthDescriptor>true</variableLengthDescriptor>
        <name>Length of Image Data</name> </field>
</x9record>

<x9record>
  <type>75</type>
  <format>0</format>
  <name>Box Summary Record</name>
  <length>f80</length>
  <allowed>false</allowed>
</x9record>

<x9record>
  <type>85</type>
  <format>0</format>
  <name>Routing Number Summary Record</name>
  <length>f80</length>
  <allowed>false</allowed>
</x9record>

</overrides>

</x9rules>
```

X9 Rules – X9Controls

X9Controls defines the high level attributes of a given x9 specification. Each attribute has a key word, allowable values, and a default value. X9Controls can be assigned at the base level and then overridden by an extension specification.

Each x9 rules definition will assign an x9 specification name that should be assigned uniquely. This name can be user defined but it is important that the name be assigned to allow the underlying x9 specification to be identified when that is desired. This is done by embedding an identifier within the x9specification name as follows:

Setting	Associated Specification
2003	This x9 specification is associated with DSTU 2003.
100-187	This x9 specification is associated with any of the x9.100-187 standards or variants.
100-187-2008	This x9 specification is associated with any of the x9.100-187-2008 standards or variants.
100-187-2013	This x9 specification is associated with any of the x9.100-187-2013 standards or variants.
100-187-2016	This x9 specification is associated with any of the x9.100-187-2016 standards or variants.
100-180	This x9 specification is associated with any of the x9.100-180 standards or variants.
100-180-2006	This x9 specification is associated with any of the x9.100-180-2006 standards or variants.
100-180-2013	This x9 specification is associated with any of the x9.100-180-2013 standards or variants.
UCD	This x9 specification is associated with any of the UCD variants.
CPA	This x9 specification is associated with CPA-015.

X9 Controls attributes are as follows:

Attribute Keyword	Usage	Values	Default
mode	Specification mode.	x9 or ach.	x9.

Attribute Keyword	Usage	Values	Default
x9Specification	The logical name assigned to this x9 specification. The associated specification should be identified as part of this name assignment. For example, a value of “MyBank 2008” would indicate that this specification is intended to be a variant of the the x9.100-187-2008 standard.	Required.	
fileNameValidation	File name validation rule to be applied to the current file name. This rule can be used in those situations where the input file name must be provided in very specific formats. This validation rule is defined in “rule/arg” format, where the rule indicates the internal rule which is to be executed and the argument is applied against that rule.	<p>“value/requiredValue” which is used when the file name is constant and must match the indicated value.</p> <p>“regex/pattern” which is used when the file name must match a provided RegEx validation pattern.</p> <p>“cpa015/” which is used when the file name must match the CPA015 pattern requirements.</p>	none
maximumFileSizeInMB	Maximum x9 file size in MB; a value of zero indicates that there is no limit.	Positive integer.	0
characterSet	Allowable character sets that can be used to express data. A given file can only utilize a single character set which is identified and assigned based on the first x9 record within	EbcDic, Ascii,either.	EbcDic

Attribute Keyword	Usage	Values	Default
	the file.		
fieldZeroPresence	Determines if field zero lengths (which identify the length of each x9 record) are required.	required, optional, prohibited.	required
fieldZeroFormat	Format used to define field zero. Note that bigEndian is the standard that has been applied to all x9 specifications.	bigEndian, littleEndian, either.	bigEndian
lineBreaks	Determines if line breaks (CRLF sequences) are allowed in NACHA files, where their presence is recognized as optional per industry standards.	required, optional, prohibited.	optional
padRecords	Determines if pad records (which appear at the end of the file) are required in NACHA files, where their presence is recognized as required per industry standards.	required, optional, prohibited.	required
dateMinimumYear	Minimum year that can appear in a date field. This validation can help to identify invalidate dates that otherwise appear valid. A value of zero disables minimum YYYY validation.	Positive integer.	0
dateMaximumYear	Maximum year that can appear in a date field. This validation can help to identify invalidate dates that otherwise appear valid. A value of zero disables maximum YYYY validation.	Positive integer.	0
dateWindowMinusDays	The number of days that are subtracted from the current date to determine the minimum	Positive integer.	0

Attribute Keyword	Usage	Values	Default
	allowable value that can be present in a date field. A value of zero disables minimum date validation.		
dateWindowPlusDays	The number of days that are added to the current date to determine the minimum allowable value that can be present in a date field. A value of zero disables maximum date validation.	Positive integer.	0
userFieldsValidated	Determines if user field validation for blanks will be applied for those x9 specifications that have defined this specific edit rule; otherwise not applicable.	true, false.	true
reservedFieldsValidated	Determines if reserved field validation for blanks will be applied for those x9 specifications that have defined this specific edit rule; otherwise not applicable.	true, false.	true
applyFieldValidations	Determines if all defined field validations within this rule set will actually be applied to the incoming x9 data. Enabling this option allows this specification to be used to test for structural integrity without actually performing individual field validations.	true, false.	true
isRequiredFieldsAreMandatory	Determines if required fields are considered as mandatory (and not conditional). Required fields are utilized by the various ACH specifications	true, false.	true

Attribute Keyword	Usage	Values	Default
	(they do not apply to x9.37).		
grayScaleImages	Determines if gray scale images are allowed as supplemental images.	true, false.	false
documentationTypeIndicatorsNoImage	Specifies the documentation type indicators that imply no image attached to the current item.	String of individual and valid documentation indicators.	"ABCDEFKL"
supplementalImageFormat	Specifies the format for supplemental images, which are those that follow the bitonal front and bitonal back images. Setting this to "any" which accept any image in any image format. Most x9 specifications will have this attribute set to none.	none, any, jpg, gif, png, tif.	none
minimumSupplementalImages	Specifies the minimum number of supplemental images that can be attached to each item. Most x9 specifications will have this attribute set to zero.	0 to 99	0
maximumSupplementalImages	Specifies the maximum number of supplemental images that can be attached to each item. Most x9 specifications will have this attribute set to zero.	0 to 99	0
multipleLogicalFilesAllowed	Determines if multiple logical files can be present within an x9 file. A logical file is the group of x9 records that begin with a file control header and end with a file trailer. This attribute is normally disabled for validation of all x9	true, false.	false

Attribute Keyword	Usage	Values	Default
	standards but may be acceptable for certain x9 variants.		
iclCollectionTypes	Defines the collection types that are used to identify an ICL file within the cash letter header.	These are defined per x9 standards but may be extended by x9 variants.	=00 01 02 12
iclRecordTypeIndicators	Defines the record type indicators that are used to identify an ICL file within the cash letter header.	These are defined per x9 standards but may be extended by x9 variants.	=E I F
iclrCollectionTypes	Defines the collection types that are used to identify an ICLR file within the cash letter header.	These are defined per x9 standards but may be extended by x9 variants.	=03 04 05 06
iclrRecordTypeIndicators	Defines the record type indicators that are used to identify an ICLR file within the cash letter header.	These are defined per x9 standards but may be extended by x9 variants.	=E I F
creditBalancing	Defines when credit balancing validation is to be applied to the current file (either x9 or ach). The “alwaysBalanced” setting indicates that credit balancing is always to be performed, where debits are to be equal to credits. The “alwaysUnbalanced” setting indicates that credit balancing is to be performed, where debits are not allowed to be equal to credits (since the processor is going to supply the needed entry). The “disabled” setting means that credit balancing is not needed and should not be performed.	alwaysBalanced, alwaysUnbalanced, disabled, mixed, deferred.	deferred

Attribute Keyword	Usage	Values	Default
	<p>The “mixed” setting indicates that credit balancing will be applied to any file that contains both debits and credits. Finally, the “deferred” setting indicates that this decision is not specified within these rules, but will instead be assigned from program options, where the default is “mixed”.</p>		
creditOrientation	<p>Defines the expected orientation for credits (deposit tickets) within a transaction set relative to the attached debits. A value of “first” indicates credits followed by debits. A value of “last” indicates debits followed by credits. A value of “any” disables this setting.</p>	first, last, any.	first
creditsCanCrossBundles	<p>Defines whether credits are allowed to cross bundles. This parameter is typically assigned a value of “true” since many institutions require a bundle size of 300 (or so) while also allowing deposits to be larger than that maximum bundle size.</p>	true, false.	true
creditsCanCrossCashLetters	<p>Defines whether credits are allowed to cross cash letters. Enabling this would be an unusual setting.</p>	true, false.	false
creditsOutOfBalanceLimit	<p>Controls the maximum number of deposit out of balance messages that will be issued.</p>		<p>Default is 10 (ten); a value of zero disables these messages.</p>
t25ClientCreditTable	<p>Defines an external table of</p>	The credit table	None; all type

Attribute Keyword	Usage	Values	Default
	type 25 routing numbers and transaction codes that are identified as user credits. The table is indexed by the ECE institution and Origination routing numbers, which allows a common table to be defined for multiple clients or applications.	name can be provided on an absolute or relative basis. If relative, then the table should be placed in / rules / x9rules /, in either the launch or home folders.	25 records are debits when this facility is not used.

X9 Rules – Basis

Basis allows an extension specification to be defined which extends another x9 specification by specifying a set of overrides to be applied to a previously defined base document. It has the following structure:

- x9rules
 - x9controls
 - basis
 - overrides
 - x9records

The following basis defines a new x9 variant which overrides another x9 specification.

```
<basis>
  <base>x9rules_dstu_2003.xml</base>
</basis>
```

An x9 variant document can be defined to override a base specification and then apply the overrides from one or more other x9 variants. When this is done, the base document is loaded and each defined extension will be applied sequentially before then applying the overrides defined within this variant.

The following is an example:

```
<basis>
  <base>x9rules_dstu_2003.xml</base>
  <extension>x9rules_frb.xml</extension>
</basis>
```

Overrides are used to modify one or more fields within an x9 record as defined within the base. Only those fields that are being overridden need to be defined within the extension. This is done as follows:

```
<x9record>
  <type>10</type>
  <format>0</format>
  <name>Cash Letter Header Record</name>
  <length>f80</length>
  <field> <item>x10.02-p003-1002-mandatory-modifiable</item>
    <edit>n</edit>
    <values>=1|2|3</values>
    <name>Collection Type Indicator</name> </field>
  <field> <item>x10.08-p043-1001-mandatory-modifiable</item>
    <edit>a</edit>
    <values>=I|E|F</values>
    <name>Cash Letter Record Type Indicator</name> </field>
</x9record>
```

Overrides can be used to turn off validation of specific fields within an x9 record as defined within the base. This is done as follows:

```
<x9record>
  <type>10</type>
  <format>0</format>
  <name>Cash Letter Header Record</name>
  <length>f80</length>
  <field> <item>x10.14-p078-1001-conditional-modifiable</item>
    <edit>a</edit>
    <validate>>false</validate>
    <name>Returns Indicator</name> </field>
</x9record>
```

Overrides can be used to indicate that a record type that is defined in the base is not validate for the extension. This is done as follows:

```
<x9record>
  <type>40</type>
```

```
<format>0</format>  
<name>Account Totals Detail Record</name>  
<length>f80</length>  
<allowed>false</allowed>  
</x9record>
```


X9 Rules – X9Record

Each x9 specification consists of a series of record definitions as documented by the underlying standard and must be defined for both base and extension XML definitions.

For a base specification, all x9 record types must be defined.

For an extension specification, only those x9 record types with differences need to be defined.

Records are defined with the following XML elements:

Xml Element	Usage	Presence	Values	Example
<type>	Record type as set by the underlying x9 specification.	Mandatory	1 through 99.	<type>01</type>
<format>	Record format is mandatory and used when a given record type has multiple defined formats. The most common usage of this functionality is the type 61 credit reconciliation record.	Mandatory	Positive integer and typically zero.	<format>0</format>
<name>	Record name.	Mandatory	Logical record name.	<name>File Header Record</name>
<length>	Defines the record format and record length for this x9 record. The record format can be either fixed (f) or variable (v). For fixed length records, the length represents that static length of the record and is typically 80 per x9 standards. For variable length records, the length field is used to identify the minimal record length that can be present.	Mandatory	“l _{nn} ” where l is either “f” or “v” and nn is the required length (fixed) and minimum length (variable).	<length>f80</length>
<location>	The location list is used to define the acceptable locations for a credit (61 or	Optional	Valid locations are as follows: a01, a10, a20,	<location>=a10 a20 a25g a61g</location>

Xml Element	Usage	Presence	Values	Example
	62)) or user (68) record within the x9 file and provides the ability to customize the validation process per your specific requirements. One or more valid record locations can be defined and are separated by the pipe character.		a25, a25a, a25g, a31, a31a, a31g, a61, a61g, a62, a62g, a70, a75, a90, any.	
creditsAddToItemCount	Determines if credits will add to the item count in all x9 trailer records.	true, false	false	<creditsAddToItemCount>true</creditsAddToItemCount>
creditsAddToTotalAmount	Determines if credits will add to the total amount in all x9 trailer records.	true, false	false	<creditsAddToTotalAmount>true</creditsAddToTotalAmount>
creditsAddToImageCount	Determines if credits will add to the image count in all x9 trailer records.	true, false	true	<creditsAddToImageCount>true</creditsAddToImageCount>
<xmlTags>	Contains five individual tags which are used to both build and parse the xml document which represents the type 68 user data field. These tags are set from an incoming string with the tags separated by the pipe character.	Optional	The xmlTags string is optional and formatted as: "xmlDocumentId fieldCountId fieldId nameId valueId".	<xmlTags>ATMUse rRecord FieldCount Field name value</xmlTags>
<field>	The list of fields that comprise this x9 record type and format.	Mandatory	Per specific and documented xml requirements.	<field> <item>x01.14-p080-l001-mandatory-modifiable</item> <edit>b</edit> <name>Reserved</name> </field>

An example of an x9 record definition is as follows:

```
<x9record>
  <type>01</type>
  <format>0</format>
  <name>File Header Record</name>
  <length>f80</length>
  <field> <item>x01.01-p001-l002-mandatory-notModifiable</item>
          <edit>n</edit>
          <name>Record Type</name> </field>
  .....
  .....
  .....
  .....
  .....
  <field> <item>x01.14-p080-l001-mandatory-modifiable</item>
          <edit>b</edit>
          <name>Reserved</name> </field>
</x9record>
```

X9 Rules – Field

Each x9 record consists of a series of field definitions. Fields are defined with the following XML elements:

Xml Element	Usage	Presence	Example
<name>	Sets the name for this field.	Mandatory	<name>Record Type</name>
<item>	Defines the high level attributes for this field.	Mandatory	<item>x01.02-p003-l002- mandatory-modifyable</ item>
	<p>Format is: xRR.FF-pPPP-LLLL- PRESENCE-MODIFYABLE</p> <p>where:</p> <p>RR is the x9 record number</p> <p>FF is the field number within the record which must be logically ascending within the definition</p> <p>LLL is the field offset within the x9 record relative to one</p> <p>PRESENCE is one of the following values:</p> <p>mandatory :: the field must be present</p> <p>conditional :: the field may be blanks but</p>		<p>Format is: xRR.FF-pPPP- LLLL-PRESENCE- MODIFYABLE</p> <p>where:</p> <p>RR is the x9 record number</p> <p>FF is the field number within the record which must be logically ascending within the definition</p> <p>LLL is the field offset within the x9 record relative to one</p> <p>PRESENCE is one of the following values:</p>

Xml Element	Usage	Presence	Example
	<p>must be valid per defined validation rules when present</p> <p>dependentMandatory :: the field is mandatory when the associated condition is met</p> <p>dependentConditional :: the field is conditional when the associated condition is met</p> <p>MODIFYABLE is one of the following values:</p> <p>modifiable :: field can be modified by X9Assist</p> <p>notModifiable :: field cannot be modified by X9Assist</p>		<p>mandatory :: the field must be present</p> <p>conditional :: the field may be blanks but must be valid per defined validation rules when present</p> <p>dependentMandatory :: the field is mandatory when the associated condition is met</p> <p>dependentConditional :: the field is conditional when the associated condition is met</p> <p>MODIFYABLE is one of the following values:</p> <p>modifiable :: field can be modified by X9Assist</p> <p>notModifiable :: field cannot be modified by X9Assist</p>
<edit>	Defines an edit rule for this field. There can be one or more edit rules assigned to each field (there is no limit as to how many can be assigned, but there is an	Mandatory	<edit>n</edit>

Xml Element	Usage	Presence	Example
	upper limit within the X9Assist field viewer. There must be at least one edit rule. If there are no logical edits to be assigned, then the “edit=none” rule should be specified.		
<validate>	Set as a convenient way to disable validation for this field without removing the edit rules. This feature is typically used to turn off validation as an override against a x9 specification when the standard validation is to be disabled.	Optional	<validate>>false</validate>
<justify>	Field justification override, which provides a value that is different than what was assigned by the edit rule. This optional feature is available for user x9 variant definitions, but there is no usage of this feature within x9 specifications provided by X9Ware. A possible usage would be to define a numeric blank field that is justified right instead of left.	Optional	<justify>right </justify>
<values>	List of valid values for this field separated by the pipe (“ ”) character.	Optional	<values>=1 2 3</values>
<iclValues>	List of possible values which is applied only when the field is within an ICL bundle.	Optional	<iclValues>01</iclValues>
<iclrValues>	List of possible values which is applied only when the field is within an ICLR bundle.	Optional	<iclrValues>01</iclrValues>
<variableLengthDescriptor>	Set when this field is a variable length descriptor for the following field. Examples are the length fields for the type 52 image reference key, digital signature, and image data.	Optional	<variableLengthDescriptor>true</variableLengthDescriptor>
<xmlid>	Set the XML ID which contains the xml field name for type 68 records when the user data field contains an xml document	Optional	<xmlid>authorizationCode</xmlid>

Xml Element	Usage	Presence	Example
	and not a structured list of individual fields.		

X9 Rules – Local Edits

Local field edits are applied by validating the actual field data against specialized rules that are based solely on inspection and do not need to take the content of other fields into consideration. Available local edits are as follows:

```
/*
 * System Local edits.
 */
b, // blanks
binary, // binary data
n, // numeric
nq, // numeric questionable
nb, // numeric blank
nbq, // numeric blank questionable
nbsm, // numeric blank special micr
ns, // numeric special
a, // alphabetic
an, // alphameric
ans, // alphameric / special
anslb, // ans allowing a leading blank (blanks can be leading and trailing)
sp, // special per x9.100-187-2013 rules
nsp, // numeric special per x9.100-187-2013 rules
ansp, // alphameric special per x9.100-187-2013 rules
ua, // upper case alphabetic
uan, // upper case alphameric
p, // alphameric with no justification per cpa005 rules
zero, // zero
nonzero, // nonzero
warnIfZero, // warn message when zero value
nonzeroByAgreementOnly, // nonzero by agreement only
min, // minimum value
max, // maximum value
none, // no validation is to be performed against this field value
sequential, // starting at 1 incremented by +1 within adjacent records of same type
sequential2, // similar to sequential, but against the first fields within the record
ascending, // generalization of sequential (no requirement on initial value or increment)
ascendingOrEqual, // generalization of ascending with duplicate values allowed
table, // table lookup using a value list located in the /rules/tables/ folder
test, // test-pass-fail using an xml testSet definition from the /rules/tests/ folder
mmyy, // date in mmyy format
mmdd, // date in mmdd format
yyymmdd, // yy >= minimum year and yy <= maximum year per current rules
yyyymmdd, // yyyy >= minimum year and yyyy <= maximum year per current rules
dateRange, // range /Low/high such as "/+0C/+3F" where C=calDays, W=weekdays, F=FRB holidays
MinimumDate, // >= Minimum date going backward x days per dateWindowMinusDays
MaximumDate, // <= Maximum date going going forward x days per dateWindowPlusDays
hhmm, // time as hhmm
hhmmss, // time as hhmmss
MICRonus, // Micr OnUs
CreditSourceOfWork, // Type 61 (DSTU) and type 62 credit source of work
ABA, // nnnnnnnnb and nnnnnnnnC (including mod10 when the check digit is present)
ABA4x4, // nnnnnnnnb, nnnnnnnnC (including mod10), and nnnn-nnnn
ABAmo10, // nine digit mod10 routing as nnnnnnnnC
AbaFileRouting, // nine digit mod10 with FRB district 00-12, 21-32, 61-72, or 5x
AbaPayorRouting, // nine digit mod10 with FRB district 00-12, 21-32, 80, or 5x
AbaEndorsementRouting, // nine digit mod10 with FRB district 00-12, 21-32, or 80
MinimumImageLength, // Minimum image length
MaximumImageLength, // Maximum image length
ImageBounds, // single rule as image bounds /MinimumImageLength/MaximumImageLength
```



```

/*
 * Validations that are used by CPA 015 and various cross border standards.
 */
CpaFileRouting, // per CPA: currency, payment type, region, site, and FI Number
AbaOrCpaFileRouting, // either nnnnnnnnC or CpaFileRouting
AbaOrCpaPayorRouting, // routing as nnnnnnnnC, nnnn-nnnn, or nnnnn-nnn,

/*
 * Validations that are unique to the East Carribbean ACH format.
 */
abaEcAch, // East Carribbean ACH has pseudo 8 digit routing with zero check digit

/*
 * Validations that are unique to ACH (NACHA).
 */
achABA8, // routing formatted as TTTTAAAA
achABA9, // routing formatted as TTTTAAAC; TTTTAAAA in this field and C in next field
achImmediateOrigin, // "bTTTTAAAC" or 10 digits numeric
achImmediateOriginNoMod10, // "bNNNNNNNNN" or 10 digits numeric
achABA10, // formatted as bTTTTAAAC (routing) or 1NNNNNNNNN (TaxID)
achEntryClass, // standard entry class
achCompanyName, // various batch header company name validations
achCompanyDesc, // various batch header company description validations

/*
 * Validations that are unique to CPA005.
 */
cp5Jdate, // six digit julian date

```

X9 Rules – Cross Field Edits

Cross field edits are applied by validating the actual field data against specialized rules that must compare the content against other fields that may exist either in this record or may exist in other records within the current x9 file. Available cross field edits are as follows:

```

/*
 * Validations that are unique to the x9 standards.
 */
CompareToFileOriginRouting, // compare to file immediate origination routing
CompareToCashLetterEceRouting, // compare to cash letter ECE institution routing
CompareToCashLetterDocType, // compare to cash letter documentation type indicator
CompareToBundleEceRouting, // compare to bundle ECE institution routing
CompareToBundleCreationDate, // compare this date to the bundle creation date
CompareToBundleBusinessDate, // compare this date to the bundle business date
CompareToBundleCycleNumber, // compare this cycle to the bundle cycle number
CashLetterIdIsUnique, // cash letter id is unique within file (FRB and CPA 015)
BundleIdIsUnique, // bundle is unique within file
CollectionTypeIndicator, // allowed values for collection type indicator
AddendumCount, // addendas attached to the current type 25/31 record
TruncIndRequiredY, // type 26/28/32/35 truncation indicator validation
TruncIndOnlyOneY, // obsolete as of R4.11
TruncIndNowhenEpcIs4, // obsolete as of R4.11
CompareToItemPayorRouting, // compare to the current item payor routing
CompareLastAddendumReturnReasonToItem, // compare last type 35 return reason to 31.6
CompareToImageDetailImageLength, // image length between type 50 and 52
CompareToImageDetailImageIndicator, // image indicator between type 50 and 52
CompareImageWidthToAuxOnUsLength, // ensure image width is sufficient for AUXOnUs chars
Icl, // allowed values when in an ICL file
Iclr, // allowed values when in an ICLR file
IclrAdmin, // allowed values when in an ICLR Administrative Returns file
IclrCustomer, // allowed values when in an ICLR Customer Returns file
MaximumCombinedImageLength, // maximum combined (front+back) image length

```

```

ViewSideIndicator, // field 50.08 view side indicator
SecurityKeySize, // field 50.12 security key size
ImagePresenceBasedOnDocType, // image present when G and not present when A-F
BlankWhenNoImage, // blank when image indicator (50.2) is '0' (no image present)
CompareToCheckDetailItemSequenceNumber, // compare to the check detail sequence number
PresenceBasedOnDigSigInd, // presence based on digital signature indicator
BundleItemCount, // bundle item count
BundleTotalAmount, // bundle total amount
BundleCreditCount, // bundle item count for x9.100-180
BundleCreditTotalAmount, // bundle total amount for x9.100-180
BundleMICRValidTotalAmount, // bundle MICR valid total amount
BundleImageCount, // bundle image count
IclBundleCount, // ICL bundle count
IclItemCount, // ICL item count
IclTotalAmount, // ICL total amount
IclCreditCount, // ICL credit count for x9.100-180
IclCreditTotalAmount, // ICL credit total amount for x9.100-180
IclImageCount, // ICL image count
CashLetterCount, // cash letter count
TotalRecordCount, // total record count
TotalItemCount, // total item count
TotalFileAmount, // total file amount
TotalFileCreditAmount, // total file credit amount

/*
 * Validations that are unique to the Canadian CPA.
 */
CpaCompareToFileHeaderOrigRouting, // compare this RT to file header origination routing
CpaOrigRoutingFormat, // validate the origin RT against the destination RT
CpaOrigRoutingFormatWhenCpa, // validate origin against destination when CPA format
MaximumAmountWhenCanadianCurrency, // maximum amount when Canadian currency
CpaValidateRoutingCurrency, // validate currency against the file header
CompareToFileHeaderCollectionType, // compare collection type CPA file header RT
CompareToCommonBusinessDate, // common business date across all cash letters
BundleSequenceIsAscending, // bundle sequence is ascending within cash letter
CashLetterDocTypeVsRecordType, // value C when 10.8 = E and G when 10.8 = I

/*
 * Validations that are unique to ACH (NACHA).
 */
AchIatCurrency, // IAT batch header currency code
AchIatRdfiBranchCountryCode, // IAT 5th IAT addenda record RDFI branch country code
AchItemAmount, // item amount rules
AchItemAddendaCount, // item addenda count
AchAddendumCompareToTrace, // compare this trace number to item trace number
AchAddendumCompare7ToTrace, // compare last 7 of this trace number to item trace number
AchValidate820Payment, // validate 820 payment data and otherwise left justification
AchBatchNumberUnique, // batch number is unique
AchServiceClass, // validate service class for this standard entry class
AchTranCodeAllowed, // validate tranCode based on entry class and service code
AchTraceCompareToBatchOdfi, // compare first 8 back to ODFI batch header
AchTraceNumber, // compare first 8 back to ODFI batch header; validate last 7 as ascending
AchCompareToServiceCode, // compare service code back to the batch header
AchCompareToBatchNumber, // compare batch number back to the batch header
AchBatchEntryAddendaCount, // bundle entry and addenda count
AchBatchHash, // bundle hash total
AchBatchDebitAmount, // bundle total debit amount
AchBatchCreditAmount, // bundle total credit amount
AchFileBatchCount, // file batch count
AchFileBlockCount, // file block count
AchFileEntryAddendaCount, // file entry and addenda count
AchFileHash, // file hash total
AchFileDebitAmount, // file total debit amount
AchFileCreditAmount, // file total credit amount

```

```

/*
 * Validations that are unique to CPA005.
 */
cp5LogicalRecordCount, // sequential position with the file
cp5OriginationControl, // match back to the file header originator + file creation number
cp5TraceNumber, // IIIICCCCFNNNNNNNN; IIII cIearId; CCCC originating data centre
// (A/U.6), FFFF file creation number (A/U.4), NNNNNNNNN item sequence number as a/n
cp5InvalidDataElementID, // formatted as AABBCDDE where first 4 groups are 04-21

```

X9 Rules – Date Range Validations

The “dateRange” local edit rule is provided to validate a date against a "low|high" date range, which is parameterized using a validation string with separate entries for the low date and high date (these two definitions are separated by the pipe character).

The basic string which describes a date within this range consists of [sign] dayCount[dayIndicator].

- The sign is either "+" or "-" which indicates the direction that the date is calculated from the current calendar date.
- The day count is the number of days to move forward or backward, depending on the sign.
- The trailing day indicator must be "C" for calendar days, "W" for week days, and "F" for work days based on the FRB holiday schedule.

This single validation rule can be applied to input dates that are either in 6 digit (YYMMDD) or 8 digit (YYYYMMDD) format.

On a Friday with a following Monday that is a FRB holiday, a value of "+0C" is calculated as Friday, "+1W" is calculated as Monday, and "+1F" is calculated as Tuesday. Similarly, a range of "+0C|+2F" would be calculated as Friday through Wednesday, and "-1C|+3F" would be calculated as Thursday through Thursday.

X9 Rules – Tables

Tables are used to define a list of values by table name. Table validations are applied by the `<edit>table/tablename</edit>` rule. Tables are useful when the same list of values are applied to numerous fields and there is a desire to only define those allowable values in a single location. Tables are also useful when the list of valid values is extremely lengthy and thus not a good candidate for direct value checking.

Tables are loaded only once for a given execution and are shared by all x9 configuration rules. All comparisons are done on a trimmed and upper case basis. Value comparisons are applied based on individual field attributions. For example, a table value of "1" when compared to a numeric field with a value of "001" will result in true.

A table example is as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<x9table>

```

```
<entry> <key>0</key> <value>Vancouver</value> </entry>
<entry> <key>1</key> <value>Montreal</value> </entry>
<entry> <key>2</key> <value>Toronto</value> </entry>
<entry> <key>3</key> <value>Halifax</value> </entry>
<entry> <key>7</key> <value>Winnipeg</value> </entry>
<entry> <key>8</key> <value>National</value> </entry>
<entry> <key>9</key> <value>Calgary</value> </entry>
</x9table>
```

X9 Rules – Tests

Test sets are applied by the `<edit>/test/testSetName</edit>` rule. Test sets are used to implement cross field validations, where the allowable content for one field is dependent on the value assigned to another field. Each test set consists of a series of tests which are applied sequentially to validate a field value. Each individual test results in a value of pass, fail, or undetermined. The execution and evaluation process for the overall test set is as follows:

- Examine and evaluate each test sequentially within the test set.
- The test set is passed when the current test entry renders an explicit result of passed.
- The test set is failed when the current test entry renders an explicit result of failed.
- Otherwise, when the current test entry renders a value of false, then the overall test set evaluation will continue with the next sequential test until all tests have been processed.
- The test set is considered as failed when all tests within the test set return a value of false.
- Each test set assigns its own unique error message to describe the field level failure, along with an optional severity level, which can be conditional based on the x9 rules used for validation.

Test sets implement an overall test-pass-fail framework where a field can be conditionally validated based on the value assigned to one or more other fields. This is the more complex condition where the allowed values for one field are dependent on the values of another field with the x9 file. This indirect reference is performed against a data record that precedes the current record position within the file. Each test definition is divided into two parts:

- "condition" which is optional and is used to associate this test to the value of another related field within the x9 file. If a condition is present, then it must be true for the attached comparison to be performed. A test is false (failed) when either the condition or the comparison is false. The condition is based on the `<field>` parameter which tags this comparison to the other related field. As indicated, the condition is optional and is defaulted to true when omitted.
- "comparison" which is optional and will be evaluated when the prefaced condition is true, or when that condition has been omitted.

The "condition" relationship is implemented as a look back from the current x9 record to preceding records within the file. This design requires that tests be defined on the second field within the relationship, so the look back (to the previous record type) can be done. This look back is limited to certain record types based on file positioning of the field that is being validated. Tests currently support look back to the following record types:

- file header;
- cash letter header;
- bundle header;
- the current item when located within an item group;
- any individual record type that is directly associated to the field which invoked the test.

A "condition" definition consists of the following components:

- The "fileSpec" parameter is used to associate this test to a specific set of x9 configurations. This relationship can be specific (eg, the 100-187-2008 specification) or more generic in nature (eg, all 100-187 specifications and their variants). This comparison is considered true when the currently loaded x9 specification contains the indicated comparison string. Typical values for this test are as follows:
 - x9.37
 - 100-187
 - 100-187-2008
 - 100-187-2013
 - 100-180
 - CPA
 - UCD
 - UCD-2008
 - UCD-2013
 - etc
- The "field" parameter is used to identify a relationship of this test to another field within the current file. The related field is identified on a record and field number basis.
- The "isEqual" or "isNotEqual" parameters which are then used to specify the required comparison for the test condition to be evaluated and determined to be successful. These comparisons can be applied to either numeric or alpha field content. The equal or not equal tests can be specified on either an absolute or indirect basis:
 - An absolute value is a single value (it cannot be a list) which identifies the required value for this test to continue. For example, <isEqual>X</isEqual>.

- An indirect value which specifies the required value based on field content. Indirect field values are identified using brackets. For example, [01.2] would reference the content of the standards level in the file control header record. A special string value [this] is used to reference the value of the field that launches the test set.
- An expression evaluation using a built-in function such as concat, substr, etc.

The "comparison" can then contain the following evaluation parameters:

- <values> which specifies a list of values which must be satisfied. The value test is defined in the same manner as the <values> parameter as used in x9 rule based tests, and can be used against both alpha and numeric fields. Alpha fields are always compared on a trimmed and upper case basis. Numeric fields are compared on a logical value basis (for example, a comparison value of "1" will be successfully matched against a field value of "001". Examples are as follows:
 - <values>=0|1|2|3</value>
 - <values>=0|1|2|20|21|22|23|24</value>
 - <values>=A|B|C|D|H|I|J|K|L|M|N|R|S|X|1|3|7|8</value>
 - <values>=//client.properties//clientRouting</value>
- <pass> which is used to explicitly pass the field value when it conforms to a data format. Allowable data formats are:
 - b (blanks)
 - p (populated)
 - n (numeric)
 - a (alpha)
 - an (alphanumeric)
 - date (YYYYMMDD format)
 - [true]
 - [false]
- <fail> which is used to explicitly fail the field value when it conforms to a data format (which is the same list as supported by <pass>).
- <minimum> which specifies a minimum value which must be satisfied. The minimum test can be performed against both numeric and alpha fields.
- <maximum> which specifies a maximum value which must be satisfied. The maximum test can be performed against both numeric and alpha fields.
- <table> which specifies that the value should be looked up in an external table. The test will be passed when the table contains the current field value. These table definitions are commonly

defined and shared with the `<edit>table/</edit>` rules, so they can be used for other validations. Tables are stored in folder `/rules/tables/`.

- `<pattern>` which specifies a RegEx pattern which must be satisfied for the test to be successful. RegEx pattern evaluations can vary considerably by environment (Java, .Net, Perl, etc) so please ensure that your pattern is written and targeted for the Java environment. There are various websites that can be very helpful in this area, which you can quickly find using a Google search. First are sites that outline the differences between the evaluation process based on platform, which helps you to understand the requirements of the Java evaluator. Second are sites that will actually perform online and interactive tests using specific patterns against specific values. We highly recommend that you use such a site during your development and testing to achieve desired results and easily determine the pass-fail results for field data combinations. Since a test set can consist of multiple tests, you can match a single field against multiple patterns. In this situation, the first test that passes will cause the overall test set to be passed.
- `<accept>` which is used to accept the field based on whether the current x9 record is located in a cash letter or bundle. This test is provided since a “condition” relationship to a field in another record will fail if that record type is determined to not exist. For example, if you do a look back to the cash letter header, and the current x9 record does not exist in a cash letter, then the conditional relationship test will fail. That result may often be desirable, but there are situations where you want to pass (and not fail) a test in that specific situation. This can be done using the accept parameter. Allowable tests are:
 - `inCashLetter`
 - `notInCashLetter`
 - `inBundle`
 - `notInBundle`
- `<message>` which contains the error message text to be issued when the test set fails.
- `<severity>` which identifies the error message severity and defaults to error. The valid error severity levels are:
 - `error`
 - `warn`
 - `info`
 - `ignore`

There are two basic formats of the error message severity tag:

- `<severity>ErrorLevel</severity>`
- `<severity>s1:e1|s2:e2|... |DefaultErrorLevel</severity>`

Where s1 (etc) is a character string that is matched against the x9 configuration name which is actively being used to validate the x9 file. If s1 is contained within the current x9 configuration name, then the e1 (etc) severity will be assigned to the created error message. Otherwise the DefaultErrorLevel will be assigned to the new error message. The s1, s2, (etc) strings are examined sequentially with the severity from the first matching string applied.

Examples of severity assignments are as follows:

- <severity>warn</severity>
- <severity>ucd:error|187:warn|ignore</severity>

In the following table, the term “expression” refers to one of the following:

- An actual string (eg, T).
- A indirect reference to a preceding field (eg, [1.10])
- An expression evaluation using a built-in function

Built-In Functions

The following built-in functions can be used to formulate values:

Built-In Function	Description	Parm1	Parm2	Parm3
substr	Substring from a defined starting position (relative to zero) for the remainder of the string.	Expression	Length	
substr	Substring from a defined starting position (relative to zero) for a defined length.	Expression	Position	Length
concat	Concatenate two or more strings, fields, or expressions.	Expression 1	Expression 2	Etc
leftPad	Pad on the left.	Expression	String	Length
rightPad	Pad on the right	Expression	String	Length
remove	Remove a string repeatedly.	Expression	String	
removeOnce	Remove a string just once.	Expression	String	
removeStart	Remove from the start of the string just once.	Expression	String	
removeEnd	Remove from the end of a string just once.	Expression	String	
removeLeading	Remove from the start of a string repeatedly.	Expression	String	
removeTrailing	Remove from the end of a string repeatedly.	Expression	String	

Built-In Function	Description	Parm1	Parm2	Parm3
repeat	Repeat a string.	Expression	String	Count
replace	Replace within a string repeatedly.	Expression	String	
replaceOnce	Replace within a string just once.	Expression	String	

External Property Files

Values can reference an externally defined value within one or more external properties files, where the property files are located in the same folder where the x9 rules are stored. The format of a properties reference is: //profileName.properties//propertyName. These external references can be used to validate specific values that may be associated with a specific client or operational area.

Example

The following is an example which uses test against to validate against a preceding field value:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Test: eexReturnReason -->
<!-- Severity: error -->
<testSet>
  <test> <field>10.14</field> <isEqual>E</isEqual> </test>
  <test> <field>10.14</field> <isEqual>R</isEqual> </test>
  <test> <field>10.14</field> <isEqual>D</isEqual> <values>=Y</values> </test>
  <test> <field>10.14</field> <isEqual>X</isEqual> <values>=Q|I|U|1|2|0</values>
    </test>
  <message>Does not match EEX cash letter returns indicator</message>
</testSet>
```

The following is an example which requires the current field value to match a preceding field value within the x9 file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Test: compareToCashLetterDestRouting -->
<!-- Severity: error -->
<testSet>
  <test> <field>10.3</field> <isEqual>[this]</isEqual> </test>
  <test> <accept>notInCashLetter</accept></test>
  <message>Must match cash letter destination routing</message>
  <repair>[10.3]</repair>
```

</testSet>

X9 Rules – POD Credit Tables

Credit tables are an advanced feature that can be used to simulate a POD capture environment. In this scenario, an Image Cash Letter (ICL) contains logical transactions which consist of credits offset by debits, where all items are identified using type 25 item detail records. Each transaction can consist of one or more credits offset by one or more debits. ICLs are often in this format, where the electronic deposit contains a deposit ticket offset by items. In fact, depending on the environment, ICLs may actually contain multiple deposits, either to the same or different accounts at the depositor's financial institution. Using this X9Assist feature adds value in several ways:

- Once a type 25 record is identified as a credit, it allows X9Assist to balance the individual transaction. A validation error will be thrown when a deposit is out of balance (the credit total amount does not equal the debit amount).
- In addition, another validation error will be thrown when the overall file is out of balance (the file credit total amount does not equal the file debit total amount).

The credit table is defined separately from x9rules. The x9controls section uses parameter "t25ClientCreditTable" to define the credit table name. This name can be provided in either an absolute or relative basis. When relative, the file must be defined within either the launch or home folder, within / rules / tables /. When absolute, the credit table can be stored within a folder that is convenient for updating. The use of an absolute location allows the definition within x9rules to be provided just once (as a pointer) with the credit table then updated externally as needed.

Credit tables can be defined using a client identifier, which is constructed from the ECE institution and destination routings from the cash letter header. This approach allows a single credit table to identify credit information for multiple clients within your capture environment.

The client table (located within the credit table) then defines one or more credits using the payor routing and transaction code from the type 25 record. Each credit configuration identifies routing(s) and transaction code(s). These are specified on a WYSIWYG basis. If both nine digit and 4-4 routings are to be accepted, then multiple routings must be provided.

For a given routing, the transaction code is optional. When the transaction code is omitted, then the routing is dedicated to credits, which means that no transaction code is required. Alternatively, the routing can be shared between debits and credits. In that situation, a credit is identified when the transaction code (from the MICR OnUs value) matches a provided value.

The client table may become a large document. X9Assist includes the Credit Table Editor which allows the credit table to be updated on an interactive basis. The credit table can also be constructed on an automated (programmatic) basis from POD (capture system) tables. Given the potential size and complexity of the credit tables, it is not recommended that they are edited using XML text editors.

Here is a sample (simplistic) credit table:

```
<?xml version="1.0" encoding="UTF-8"?>
<creditTable>
```

```

<clients>
  <client>
    <eceRouting>123456780</eceRouting>
    <destRouting>123456780</destRouting>
    <description/>
    <credit>
      <routing>555555550</routing>
      <tranCode>5</tranCode>
      <tranCode>05</tranCode>
      <tranCode>005</tranCode>
    </credit>
  </client>
</clients>
</creditTable>

```

Here is a sample x9rules definition that is an extension to a basis document and includes definition of the credit table. This sample first defines the credit table and then provides an override for the type 25 record to indicate that the type 25 credit counts and amounts are to be added into trailer totals (note that this is also the default, so this entire type 25 override could be omitted). However, in some environments, it may be desirable to include this with other settings (for example, add type 25 credit counts into trailer totals). The x9rules sample is as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<x9rules>
  <x9Controls>
    <x9Specification>Test Rules x9.100-187-2008</x9Specification>
    <creditOrientation>first</creditOrientation>
    <creditsCanCrossBundles>true</creditsCanCrossBundles>
    <creditsCanCrossCashLetters>>false</creditsCanCrossCashLetters>
    <t25ClientCreditTable>sampleCreditTable.xml</t25ClientCreditTable>
  </x9Controls>
  <basis>
    <base>x9rules_x9.100-187.xml</base>
  </basis>
  <overrides>
    <x9record>
      <type>25</type>
      <format>0</format>
      <length>f80</length>
      <creditsAddToItemCount>true</creditsAddToItemCount>
      <creditsAddToTotalAmount>true</creditsAddToTotalAmount>
      <name>Check Detail Record</name>
    </x9record>
  </overrides>
</x9rules>

```

TIFF Rules

TIFF image validation is a complex part but also very important facet of the x9 file validation process. X9Ware has a very powerful TIFF validation engine that is XML driven, allowing the validation process to be customized per specific user requirements.

The TIFF tag XML document is the core structure which defines TIFF validation for a given x9 specification. It has the following basic structure:

- tiffrules
 - tiffControls
 - tiffEdits
 - tiffEdit
 - mandatoryTiffTags
 - tiffTagDescriptions

It is also possible to define separate TIFF validation rules for black white versus gray scale images. When using this functionality there can be up to four 50-52-54 image sequences, where the first image set is associated with the black white image and the second image set is associated with the gray scale image.

The following XML structure is used when defining both black white and gray scale rules:

- tiffrules
 - blackWhite
 - tiffControls
 - tiffEdits
 - tiffEdit
 - mandatoryTiffTags
 - grayScale
 - tiffControls
 - tiffEdits
 - tiffEdit
 - mandatoryTiffTags
 - tiffTagDescriptions

TIFF Rules – TIFF Controls

TIFF Controls are used to define the high level attributes of a given TIFF validation set. Each attribute has a key word, allowable values, and a default value.

TIFF Controls attributes are as follows:

Attribute Keyword	Usage	Values	Default
<endianFormat>	Defines whether the IFD is must be little endian or big endian. In the “II” format, byte order is always from the least significant byte to the most significant byte, for both 16-bit and 32-bit integers This is called little-endian byte order. In the “MM” format, byte order is always from most significant to least significant, for both 16-bit and 32-bit integers. This is called big-endian byte order.	littleEndian, bigEndian, either.	littleEndian.
<ifdOnWordBoundary>	Defines whether the IFD must begin on a word boundary.	true, false.	false.
<ascendingTagsRequired>	Defines if tags must be in numerically ascending sequence.	true, false.	true.
<duplicateTagsAllowed>	Defines if duplicate tags are allowed.	true, false. false.	false.
<privateTagsAllowed>	Defines if private tags are allowed.	true, false.	true.

Attribute Keyword	Usage	Values	Default
<privateDuplicateTagsAllowed>	Defines if private duplicate tags are allowed.	true, false.	false.
<multiStripAllowed>	Defines if multi-strip images will be allowed.	true, false.	true.
<validateEOFB>	Defines if EOFB validation will be performed.	true, false.	true.
<maximumEofbZeroPadBytes>	Maximum number of EOFB zero pad bytes which can be appended to the end of the standard EOFB sequence. Per Group4 Fax standards, this should be set to zero. However, some TIFF encoders will append pad bytes and this parameter allows these images to be accepted without error.	Positive integer.	0.
<imageMinWidth>	Minimum image width defined in inches.	Decimal value.	4.100.
<imageMaxWidth>	Maximum image width defined in inches.	Decimal value.	10.500.
<imageMinHeight>	Minimum image height defined in inches.	Decimal value.	1.752.
<imageMaxHeight>	Maximum image height defined in inches.	Decimal value.	5.700.
<applyIqaRules>	Defines if IQA rules will be applied as part of x9 file validation.	true, false.	false.
<applyIqaToFrontOnly>	Defines if IQA rules will	true, false. true.	true.

Attribute Keyword	Usage	Values	Default
	be applied only to front images.		
<frontTooLight>	Minimum percentage of black pixels which must be present. Minimum percentage of black pixels which must be present.	Percent of pixels (from 0 to 100 expressed as a decimal value).	0.900
<frontTooDark>	Maximum percentage of black pixels which must be present.	Percent of pixels (from 0 to 100 expressed as a decimal value).	90.000
<backTooLight>	Minimum percentage of black pixels which must be present.	Percent of pixels (from 0 to 100 expressed as a decimal value).	0.380
<backTooDark>	Maximum percentage of black pixels which must be present.	Percent of pixels (from 0 to 100 expressed as a decimal value).	98.000

An example is as follows:

```

<tiffControls>
  <endianFormat>either</endianFormat>
  <ifdOnWordBoundary>>false</ifdOnWordBoundary>
  <ascendingTagsRequired>>true</ascendingTagsRequired>
  <duplicateTagsAllowed>>false</duplicateTagsAllowed>
  <privateTagsAllowed>>true</privateTagsAllowed>
  <privateDuplicateTagsAllowed>>false</privateDuplicateTagsAllowed>
  <multiStripAllowed>>true</multiStripAllowed>
  <validateEOFB>>false</validateEOFB>
  <maximumEofbZeroPadBytes>0</maximumEofbZeroPadBytes>

```

```

<imageMinWidth>4.100</imageMinWidth>
<imageMaxWidth>10.500</imageMaxWidth>
<imageMinHeight>1.752</imageMinHeight>
<imageMaxHeight>5.700</imageMaxHeight>
<applyIqaRules>>false</applyIqaRules>
<applyIqaToFrontOnly>>true</applyIqaToFrontOnly>
<frontTooLight>0.900</frontTooLight>
<frontTooDark>90.000</frontTooDark>
<backTooLight>0.380</backTooLight>
<backTooDark>98.000</backTooDark>
</tiffControls>

```

TIFF Rules – TIFF Edits

TIFF Edits are used to define the validation attributes for a given TIFF tag. Allowable attributes are as follows:

Xml Element	Usage	Presence	Values	Example
<tag>	Tiff tag number.	Mandatory	1 through 65535.	<tag>266</tag>
<type>	Type defines the field type per the TIFF 6.0 standard.	Mandatory	BYTE, ASCII, SHORT, LONG, RATIONAL, SBYTE, UNDEFINED, SSHORT, SLONG, SRATIONAL, FLOAT, DOUBLE.	<type>LONG</type>
<count>	Count s the number of values that re present in the directory entry. Note that Count is not the total number of bytes. For example, a single 16-bit word (SHORT) has a Count of 1 and not 2.	Mandatory	Positive integer.	<count>1</count>

Xml Element	Usage	Presence	Values	Example
<values>	Defines a list of acceptable values.	Optional	List of values separated by the pipe character.	<values>=200 240</values>
<variance>	Defines a list of values that will be reported on a variance basis.	Optional	List of values separated by the pipe character.	
<warn>	Defines a list of values that will be reported on a warning basis.	Optional	List of values separated by the pipe character.	
<info>	Defines a list of values that will be reported on an info basis.	Optional	List of values separated by the pipe character.	
<minimum>	Defines a minimum value.	Optional	Positive integer.	<minimum>2</minimum>
<maximum>	Defines a maximum value.	Optional	Positive integer.	<maximum>2</maximum>
<rule>	Defines specialized rules that can be applied to TIFF tag validation.	Optional	nonzero notAllowed notAllowedWarn notAllowedInfo tag278RowsPerStrip	<rule>notAllowed</rule>

An example is as follows:

```

<tiffEdit>
  <tag>266</tag>
  <type>SHORT</type>
  <count>1</count>
  <values>=1|2</values>
  <variance>=2</variance>
</tiffEdit>

```

TIFF Rules – Mandatory TIFF Tags

This XML element provides a list of all TIFF tags which are considered as mandatory. An error will be thrown if the image does not include this tag.

An example is as follows:

```
<mandatoryTiffTags>  
  <tag>256</tag>  
  <tag>257</tag>  
  <tag>259</tag>  
  <tag>262</tag>  
  <tag>273</tag>  
  <tag>278</tag>  
  <tag>279</tag>  
  <tag>282</tag>  
  <tag>283</tag>  
</mandatoryTiffTags>
```

TIFF Rules – TIFF Tag Descriptions

The tag description list is used to provide overrides to the standard list of tiff tag descriptions that are by default included in our tiff rules support. The standard descriptions cover all tiff tags which are commonly present in a typical x9.100-181 black-white image that is valid for image exchange, and also includes tiff tags as defined by the TIFF 6.0 standard. We will gladly add common tiff tags to our standard list if they are generally used through the industry. Please send them to X9Ware and they will be incorporated.

This tag list is defined within the tiff rules and can contain either additions or replacements to the standard list. Each entry contains the numeric tag value and the associated string name. The tag description will be reported as unknown when not defined within this list.

```
<tiffTagDescriptions>  
  <tiffTag> <tag>254</tag> <description>NewSubfileType</description> </tiffTag>  
</tiffTagDescriptions>
```

Gray Scale Image Support

Gray scale images are implemented through the x9 and tiff validation rules. Typically, image exchange supports two bitonal images (front and back). The addition of gray scale allows you to have three or four images. The first two images are bitonal front and back. This is followed by the gray scale front and then the gray scale back.

X9 Rules

The required presence of the gray scale images is based on upon the “grayScaleImages” parameter within your x9 rules definition. This value must be defined as follows:

none	The x9 file will have bitonal images only; gray scale images are not allowed.
front	Each item must have a single gray scale image which is the front image. This implies that each item will have three 50/52 sequences.
front_back	Each item must have two gray scale images which are the front and back images respectively. This implies that each item will have four 50/52 sequences.
optional	Each item may (or may not) have gray scale images. Each item must have a minimum of two and a maximum of four 50/52 sequences. The gray scale front image must be in relative position three when present. The gray scale back image must be in relative position four when present.

Please reference our x9rules_CPA_GrayScale.xml definition for a sample as to how to enable gray scale image validation as x9 rules variant.

```
<?xml version="1.0" encoding="UTF-8"?>
<x9rules>

  <x9Controls>
    <x9Specification>CPA 015 GrayScale</x9Specification>
    <characterSet>EbcDic</characterSet>
    <maximumChecksPerFile>40000</maximumChecksPerFile>
    <fieldZeroPresence>required</fieldZeroPresence>
    <fieldZeroFormat>bigEndian</fieldZeroFormat>
    <dateMinimumYear>2000</dateMinimumYear>
    <dateMaximumYear>2099</dateMaximumYear>
    <dateWindowMinusDays>1095</dateWindowMinusDays>
    <dateWindowPlusDays>95</dateWindowPlusDays>
    <userFieldsValidated>false</userFieldsValidated>
```

```

    <reservedFieldsValidated>>false</reservedFieldsValidated>
    <grayScaleImages>front_back</grayScaleImages>
    <multipleLogicalFilesAllowed>>false</multipleLogicalFilesAllowed>
    <iclCollectionTypes>=01</iclCollectionTypes>
    <iclRecordTypeIndicators>=E|I</iclRecordTypeIndicators>
    <iclrCollectionTypes>=03|04|05|06</iclrCollectionTypes>
    <iclrRecordTypeIndicators>=E|I|F</iclrRecordTypeIndicators>
</x9Controls>

<basis>
  <base>x9rules_x9.100-187_CCD.xml</base>
</basis>

<overrides>
</overrides>

</x9rules>

```

TIFF Rules

TIFF validation just include separate sections that are used to validate TIFF tags within bitonal versus gray scale images. Please reference our tiffrules_GrayScale.xml definition for a sample as to how to define gray scale image validation within our tiff rules.

```

<?xml version="1.0" encoding="UTF-8"?>
<tiffrules>

  <tiffControls>
    <endianFormat>either</endianFormat>
    <ifdOnWordBoundary>>false</ifdOnWordBoundary>
    <ascendingTagsRequired>>true</ascendingTagsRequired>
    <duplicateTagsAllowed>>false</duplicateTagsAllowed>
    <privateTagsAllowed>>true</privateTagsAllowed>
    <privateDuplicateTagsAllowed>>false</privateDuplicateTagsAllowed>
    <multiStripAllowed>>true</multiStripAllowed>
    <validateEOFB>>false</validateEOFB>
    <maximumEofbZeroPadBytes>0</maximumEofbZeroPadBytes>
    <imageMinWidth>4.100</imageMinWidth>
    <imageMaxWidth>10.500</imageMaxWidth>
    <imageMinHeight>1.752</imageMinHeight>
    <imageMaxHeight>5.700</imageMaxHeight>
    <applyIqaRules>>false</applyIqaRules>
    <applyIqaToFrontOnly>>true</applyIqaToFrontOnly>
    <frontTooLight>2.660</frontTooLight>
    <frontTooDark>90.000</frontTooDark>
    <backTooLight>0.380</backTooLight>
    <backTooDark>90.000</backTooDark>
  </tiffControls>

```

</tiffControls>

<tiffEdits>

<tiffEdit>

<tag>254</tag>

<type>LONG</type>

<count>1</count>

<values>=0</values>

</tiffEdit>

<tiffEdit>

<tag>255</tag>

<type>SHORT</type>

<count>1</count>

<values>=1</values>

</tiffEdit>

<tiffEdit>

<tag>256</tag>

<type>SHORT_LONG</type>

<count>1</count>

<rule>nonzero</rule>

</tiffEdit>

<tiffEdit>

<tag>257</tag>

<type>SHORT_LONG</type>

<count>1</count>

</tiffEdit>

<tiffEdit>

<tag>258</tag>

<type>SHORT</type>

<count>1</count>

<values>=1</values>

</tiffEdit>

<tiffEdit>

<tag>259</tag>

<type>SHORT</type>

<count>1</count>

<values>=4</values>

</tiffEdit>

<tiffEdit>

<tag>262</tag>

<type>SHORT</type>

<count>1</count>

<maximum>1</maximum>

</tiffEdit>

<tiffEdit>

<tag>263</tag>

<type>SHORT</type>

<count>1</count>

<values>=1</values>

</tiffEdit>

```
<tiffEdit>
  <tag>266</tag>
  <type>SHORT</type>
  <count>1</count>
  <values>=1|2</values>
  <variance>=2</variance>
</tiffEdit>
<tiffEdit>
  <tag>273</tag>
  <type>SHORT_LONG</type>
</tiffEdit>
<tiffEdit>
  <tag>274</tag>
  <type>SHORT</type>
  <count>1</count>
  <values>=1</values>
</tiffEdit>
<tiffEdit>
  <tag>277</tag>
  <type>SHORT</type>
  <count>1</count>
  <values>=1</values>
</tiffEdit>
<tiffEdit>
  <tag>278</tag>
  <type>SHORT_LONG</type>
  <count>1</count>
  <rule>tag278RowsPerStrip</rule>
</tiffEdit>
<tiffEdit>
  <tag>279</tag>
  <type>SHORT_LONG</type>
</tiffEdit>
<tiffEdit>
  <tag>282</tag>
  <type>RATIONAL</type>
  <count>1</count>
  <values>=200|240</values>
</tiffEdit>
<tiffEdit>
  <tag>283</tag>
  <type>RATIONAL</type>
  <count>1</count>
  <values>=200|240</values>
</tiffEdit>
<tiffEdit>
  <tag>284</tag>
  <type>SHORT</type>
  <count>1</count>
</tiffEdit>
```

```

    <tiffEdit>
      <tag>293</tag>
      <type>LONG</type>
      <count>1</count>
      <values>=0</values>
    </tiffEdit>
    <tiffEdit>
      <tag>296</tag>
      <type>SHORT</type>
      <count>1</count>
      <values>=2</values>
    </tiffEdit>
    <tiffEdit>
      <tag>320</tag>
      <type>SHORT</type>
      <rule>notAllowed</rule>
    </tiffEdit>
    <tiffEdit>
      <tag>321</tag>
      <type>SHORT</type>
      <rule>notAllowed</rule>
    </tiffEdit>
  </tiffEdits>

  <mandatoryTiffTags>
    <tag>256</tag>
    <tag>257</tag>
    <tag>259</tag>
    <tag>262</tag>
    <tag>273</tag>
    <tag>278</tag>
    <tag>279</tag>
    <tag>282</tag>
    <tag>283</tag>
  </mandatoryTiffTags>

  <tiffTagDescriptions>
    <!--
  <tiffTag> <tag>254</tag> <description>NewSubfileType</description> </tiffTag>
  -->
  </tiffTagDescriptions>

</tiffrules>

```


X9 Messages

Error messages can become a complex topic, especially when there is a need to dramatically customize them on a case-by-case basis. You will need to read this topic in its entirety to fully understand all of available options.

The message system itself is designed in a manner that allows it to be manipulated by X9Assist users, through the Message Editor and Configuration Editor, which are UI editors that manipulate the various xml files that are used to control the underlying error messaging process.

Although X9Ware-SDK users can leverage these editors to control the error messaging subsystem, they can also use the X9Ware-SDK API to more fully control error messages within the runtime environment. This direct use of message assignments, via the X9Ware-SDK API, can be more straight forward way to achieve specific needs.

Message XML

The messages XML document defines all errors that can be originated within the environment. Each message has a logical name and an associated severity and description. Messages are defined with the following XML elements:

Xml Element	Usage	Presence	Values	Example
<id>	ID is used to logically identify each error message.	Mandatory	Internally assigned.	<id>bindFailure</id>
<sev>	Assigns the severity associated with this error. The available values are: Severe, Error, Warn, Info, and Ignore. Messages that are assigned as Ignore are not reported as errors.	Mandatory	Severe Error Warn Info Ignore	<sev>Error</sev>
<desc>	Error description which is the text that is inserted into the error message for this error condition.	Mandatory	Must be defined for each error.	<desc>configuration bind failure</desc>
<recordType>	Record type for this error message. There can be multiple definitions for the same message, with different text applied to various record types.	Optional	The relevant record type.	<recordType>1</recordType>
<fieldName>	Field name for this error message. Record type is required when using field name. There can be multiple definitions for the same message, with different text applied to various record types / field name combinations.	Optional	The relevant field name.	<fieldName>Immediate Origin</fieldName>
<pattern>	The message pattern that will be applied to this message.	Optional	See the pattern topic (below) for more information as to how this string is defined.	
<format>	Additional information as to how the message should be formatted. These values provide further direction as to how fields are populated within the pattern. A value of “full” indicates that all fields (that are present) will	Optional	Full Sparse Plain	<format>Plain</format>

Xml Element	Usage	Presence	Values	Example
	be populated, while a value of “plain” indicates that the message will be limited to just the message text itself. A value of “sparse” indicates that the message text and comments will be included. Default is full.			

An example is as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<x9errors>
<error> <id>bindFailure</id> <sev>Error</sev>
      <desc>configuration bind failure</desc>
</error>
.....
.....
</x9errors>

```

System Messages

System messages are stored in the “systemMessages.xml” file which is stored within the X9Ware-SDK JAR as embedded resources, in location / resources / rules / messages /. These are the standard system messages, and are designed to be utilized as distributed. The X9Ware-SDK expects these messages to remain as-is, and not modified to meet specific user objectives.

Override Messages

Override message files are stored in separate xml files and are intended to contain user message overrides, where those assignments are different than the standard messages that are stored in the “systemMessages.xml” definition. An override message file is applied on top of the system messages themselves, which means that they can either replace or add to that core message set. Using override messages is an important part of our application design, since it allows your overrides to be independent of those messages that are distributed with the X9Ware-SDK. This has several benefits. First is that you can easily manage your overrides, since they are isolated and separately defined. Second is that this simplifies the process of going to a new release, since you do not need to be as concerned about messages that have been added or removed with a new X9Ware-SDK code base.

Full / Sparse / Plain

Each message has an assigned format, which is used to control the fields that are attached to the message via the pattern. The message format is provided for convenience, since it can be used to limit

the fields that are attached to a message without the need to assign a unique pattern. Format must be one of the following:

- Full – messages will have all available fields (those that are non-blank) attached to the resulting message, per the message pattern.
- Sparse – messages will have a reduced set of fields assigned, in the interest of brevity.
- Plain – messages will contain the text only.

Message Patterns

Message patterns are used to define how error messages are formatted. The patterns contain both static strings (which are copied directly into the constructed messages) as well as replacement fields (which are substituted with data from the current error being thrown).

Message patterns can be defined at multiple levels within the environment:

- Global – the global message pattern is applied to all messages that are not assigned a pattern through one of the lower levels.
- Specific error identifier – the pattern is assigned for a specific message identifier.
- Specific error identifier and record type – the pattern is assigned for a specific message identifier and record type.
- Specific error identifier, record type, and field name – the pattern is assigned for a specific message identifier, record type, and field name.

Fields that are available for building a message pattern are defined in X9MessageManager as public, which allows an application program to use these strings to build a custom pattern, for those situations where the standard pattern must be changed.

The “|” character is used to as a separator to divide the message pattern into a series of fields. Each field is included in the constructed message when the underlying data is present, but is omitted when the data for that particular fields is not present. The use of each supported field type is optional. You can omit a particular field when it is not desirable. The pattern also controls the order that is used to formulate the resulting error message. Through this facility, you have complete control over the fields that are used to build an error message and the order that message is constructed.

Here is an example of how a message pattern might be defined:

```
|{Prefix}|{Line}|field {RecordDotField}|{Name}|position {Position}|{Description}|value  
'{FieldValue}'expecting '{ExpectedValue}()'|{Comments}|failed {Reason}|[{Sev}]|
```

Within the X9Ware-SDK, the API can be used to define a message pattern. Here is the standard message pattern as defined in X9MessageManager:

```
public static final String DEFAULT_MESSAGE_PATTERN = ESEPARATOR + PREFIX +  
ESEPARATOR + LINE_NUMBER + ESEPARATOR + "field " +  
RECORD_DOT_FIELD + ESEPARATOR + FIELD_NAME + ESEPARATOR +
```

```
"position " + FIELD_POSITION + ESEPARATOR + ERROR_DESCRIPTION +
ESEPARATOR + "value " + QUOTE_MARK + FIELD_VALUE + QUOTE_MARK +
ESEPARATOR + "expecting " + QUOTE_MARK + EXPECTED_VALUE +
QUOTE_MARK + ESEPARATOR + ERROR_COMMENTS + ESEPARATOR +
"failed " + FAILED_REASON + ESEPARATOR + "[" + SEVERITY + "]" +
ESEPARATOR;
```

Here are the static strings that can be used to define a custom message pattern:

```
public static final String PREFIX = "{Prefix}"; // typically "record" or "line"
public static final String LINE_NUMBER = "{Line}";
public static final String RECORD_DOT_FIELD = "{RecordDotField}";
public static final String RECORD_TYPE = "{Record}";
public static final String FIELD_NUMBER = "{Field}";
public static final String FIELD_NAME = "{Name}";
public static final String FIELD_POSITION = "{Position}";
public static final String FIELD_START = "{Start}";
public static final String FIELD_END = "{End}";
public static final String FIELD_LENGTH = "{Length}";
public static final String FIELD_VALUE = "{FieldValue}";
public static final String EXPECTED_VALUE = "{ExpectedValue}()";
public static final String ERROR_DESCRIPTION = "{Description}";
public static final String ERROR_COMMENTS = "{Comments}"; // includes value ??
public static final String ERROR_NAME = "{Name}";
public static final String FAILED_REASON = "{Reason}";
public static final String SEVERITY = "{Sev}";
public static final String SEVERITY_WARN_OR_LESS = "{SevWarnOrLess}";
```

Message Files in JAR versus File System

Messages can be defined in one of two locations.

- First is the within the X9Ware-SDK JAR, as part of the resources bundle. To include message files in the JAR, users must unzip the JAR itself, add the new or modified XML file into the appropriate folder location, and then use a compatible ZIP process to rebuild the JAR. You can do some internet searches to find good examples of the tools that can be used to perform these tasks in a manner that is compatible with the JVM class loader.
- Second is within the external file system, where XML files can be stored in a similar folder structure, just externally from the X9Ware-SDK JAR. This approach has both advantages and disadvantages. On the plus side, the XML components can be quickly and easily updated. However, storing these separately, can be negative in that it complicates the overall runtime environment.

Message Pattern Reuse

Within the message XML definitions, a message pattern can be assigned a logical name when it is initially defined, and then that name can be used to subsequently reference the pattern. This approach is very beneficial when the same message pattern is going to be repetitively reused. It allows the message

pattern to be defined just once. This creates a single version of the pattern, which simplifies both definition and ongoing maintenance.

The message pattern reuse facility is leveraged by assigning the logical name when it is first used, and then using that name as proxy for the pattern when subsequently used.

You can define different message patterns to meet different error message requirements. You would then assign a logical name to each pattern which describes the intent of the message pattern within your application.

The pattern must be defined before it can be referenced. The initial definition will look like `<pattern>patternName=|patternString|</pattern>`, while subsequent references will then look like `<pattern>patternName</pattern>`.

Message Configurations

Before running a file validation, the X9Ware-SDK requires that the application first bind to a configuration. Each such configuration is associated with a set of xml rules and as well as a set of xml messages. IN this fashion, users can define their own configurations, where a configuration can be associated with their own custom set of error messages where these xml messages are actually overrides that sit on top of the standard message file.

X9Ware-SDK applications are required to bind to a configuration, using code something as follows:

```
if (!sdkBase.bindConfiguration(X9.ACH_CORE_VALIDATIONS_CONFIG)) {  
    throw X9Exception.abort("bind unsuccessful");  
}
```

The Configuration Editor (available within X9Assist) can be used to update the config.xml file, which is where configurations are defined. A given environment can define one or more configurations that are additions to the standard configuration set.

In addition to using the Configuration Editor, X9Ware-SDK applications can also use the API to insert configurations directly into the environment, thus eliminating the need to manipulate config.xml itself. This is done using X9ConfigManager and addMapEntry().

Using the Message and Configuration Editors

The X9Assist Message Editor is used to maintain a message xml file that is an override to the standard system messages. Individual message xml entries are saved only when they are different from the standard system definition. The Message Editor panel allows you to enter these and save them to an xml file of your choice. The resulting messages xml file can then be inserted into the JAR or referenced externally in the file system. These message xml files must be associated within a configuration definition to be activated.

The X9Assist Configuration Editor is used to maintain a configuration xml file that contains user entries, which are an extension to the standard system entries. The resulting xml file is saved as config.xml by the editor.

Using the X9Ware-SDK API to Insert Message Overrides

X9Ware-SDK applications can use the X9Ware-SDK API to dynamically insert messages into the runtime environment. This direct approach can perhaps be the most straight forward when using the SDK. Unique patterns can be defined and assigned to messages as required, allowing the messages to be very targeted to specific error situations. Also note that message patterns are ready-only (they are not modified), so the same pattern can be shared with multiple message definitions. Sample code to accomplish this (using the X9Ware-SDK API) is as follows:

```
/*
 * Store override messages into the message manager lookup map. This runtime map is
 * initially populated based on the currently loaded configuration, which can now be
 * further manipulated. These overrides can be either replacements for the standard
 * message, or additions at the record or record/field levels. Each override is
 * associated with the X9Msg enumerator that is related to the message definition in the
 * systemMessages.xml definition, which defines the standard message that would
 * otherwise be issued on a default basis. Each override contains the new message to be
 * issued and the severity level to be assigned. Overrides can be global (applied to all
 * such messages) or can be tied to a specific record type and optional field name.
 *
 */
final X9MessageManager x9messageManager = sdkBase.getMessageManager();
x9messageManager.putMessage( // override using text/recordType/severity
    new X9Message(X9Msg.lessThanMinimum, "less than minimum date",
        X9Ach.BATCH_HEADER, "EFFECTIVE ENTRY DATE",
        X9C.SEVERITY_WARN));
x9messageManager.putMessage( // override using text/recordType/fieldName/severity
    new X9Message(X9Msg.mandatory, "company identifier is missing",
        X9Ach.BATCH_CONTROL, "COMPANY IDENTIFICATION",
        X9C.SEVERITY_WARN));

/*
 * Store an even more complex override message that includes an alternate pattern, which
 * will be used to format this specific error.
 */
final char separator = X9MessageManager.ESEPARATOR;
final String overridePattern = separator + X9MessageManager.PREFIX + separator
    + X9MessageManager.LINE_NUMBER + separator + "field "
    + X9MessageManager.FIELD_POSITION + separator
    + X9MessageManager.ERROR_DESCRIPTION + separator + "["
    + X9MessageManager.SEVERITY_WARN_OR_LESS + "]" + separator;
x9messageManager.putMessage(
    new X9Message(X9Msg.achIncorrectHash, "file hash total is incorrect",
        X9Ach.FILE_CONTROL, "ENTRY HASH", overridePattern,
        X9C.SEVERITY_ERROR));
```

Using the X9Ware-SDK API to Insert Message Overrides from XML

X9Ware-SDK applications can also package their message overrides within an override xml file which can then be used as an overlay on top of systemMessages.xml. The xml file can be built using our Message Editor, or can be constructed through other means such as an xml or text editor.

The above example shows three messages being inserted individually into the runtime environment using the X9Ware-SDK API. These messages could instead be defined as an xml file, which would be structured as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<errorMessages>
  <copyright>X9Ware LLC 2012-2018</copyright>
  <company>X9Ware LLC</company>
  <release>R4.06</release>
  <buildDate>2021.03.04</buildDate>
  <timestamp>20210304_090217_975</timestamp>
  <basis></basis>
  <pattern>[default]</pattern>
  <messages>
    <error>
      <id>achIncorrectHash</id>
      <sev>Error</sev>
      <desc>File hash total is incorrect</desc>
      <recordType>9</recordType>
      <fieldName>Entry Hash</fieldName>
      <pattern></pattern>
      <format>Full</format>
      <type>User</type>
    </error>
    <error>
      <id>lessThanMinimum</id>
      <sev>Warn</sev>
      <desc>Less than allowed minimum value</desc>
      <recordType>0</recordType>
      <fieldName></fieldName>
      <pattern></pattern>
      <format>Full</format>
      <type>User</type>
    </error>
    <error>
      <id>mandatory</id>
      <sev>Warn</sev>
      <desc>Company identifier is missing</desc>
      <recordType>8</recordType>
      <fieldName>Company Identification</fieldName>
```



```

    <pattern>|{Prefix}|{Line}|field {Position}|{Description}|[{SevWarnOrLess}]|</pattern>
    <format>Full</format>
    <type>User</type>
  </error>
</messages>
</errorMessages>

```

The xml file can then be defined either externally (in runtime folder / rules / messages /) or internally as a resource (in jar folder / rules / messages /). From either of these locations, the message override file can be loaded using the message manager, as follows:

```

try {
    final X9MessageManager x9messageManager = sdkBase.getMessageManager();
    x9messageManager.loadResourceFile("messageOverrides.xml");
} catch (final Exception ex) {
    throw X9Exception.abort(ex);
}

```

In this situation, the following logging is issued when the standard system messages are loaded:

```

2021-03-04 09:40:43.028 [INFO] xml loaded(systemMessages.xml) isContentLogging(false)
class(com.x9ware.beans.X9MessageBean) from(com.x9ware.jaxb.X9Jaxb.logUnmarshalledResults:522)
(com.x9ware.tools.X9MiniLog.writeToLog:472)
2021-03-04 09:40:43.091 [INFO] xml messageCount(272) fileName(systemMessages.xml)
systemMessageCount(272) overrideMessageCount(0) testsetMessageCount(0)
(com.x9ware.messaging.X9MessageManager.loadResourceFile:285)

```

Alternatively the following logging is issued when the override messages are loaded. This logging shows the override message count as two, since one of the messages replaced a standard message and the other two were net additions.

```

2021-03-04 09:40:43.216 [INFO] xml loaded(messageOverrides.xml) isContentLogging(false)
class(com.x9ware.beans.X9MessageBean) from(com.x9ware.jaxb.X9Jaxb.logUnmarshalledResults:522)
(com.x9ware.tools.X9MiniLog.writeToLog:472)
2021-03-04 09:40:43.218 [INFO] xml messageCount(274) fileName(messageOverrides.xml)
systemMessageCount(272) overrideMessageCount(2) testsetMessageCount(0)
(com.x9ware.messaging.X9MessageManager.loadResourceFile:285)

```

Bitonal Image Thresholding

Bitonal image thresholding is the process used to convert gray scale images into binary images, where each pixel is classified as either black or white based on its intensity value. There are a variety of imaging algorithms to accomplish this, where each of these methods utilize their own core process to identify one or more threshold values. There can be a single threshold that applies to the entire image, or the algorithms can be a more complex where there are multiple thresholds, where each is adapted to the local area within the image. Pixels with intensities above the threshold are assigned to one class (usually white), while pixels with intensities below or equal to the threshold are assigned to the other class (typically black).

Bitonal Image Challenges

Significant challenges exist in this process. Gray scale images may contain noise that can affect the accuracy and the output image that is created by the thresholding process. To address this, pre-processing steps like smoothing or filtering can be applied to reduce noise before applying the thresholding. Another challenge arises from a multitude of issues with the input image itself. This can be caused by scanner noise, image artwork, complex backgrounds, camera problems with mobile devices, and varying lighting conditions when the image is captured. All of these can lead to uneven intensity values, which can result in undesirable results where images are washed out to black making the output image unusable. Adaptive thresholding methods attempt to mitigate this issue by adjusting the threshold locally based on image content can mitigate this issue.

Within our SDK and X9Utilities, we have implemented a number of thresholding methods that are sequentially, in an attempt to generate a usable output image despite initial image capture issues. This is accomplished by applying a variety of thresholding techniques and evaluation the resulting image for usability. This process ultimately selects the image that, based on our inspection, appears to provide the most usable output image.

Bitonal Thresholding Techniques

Our thresholding process first invokes the standard Java ImageIO conversion from gray scale to bitonal that is provided by the JDK. This result is accepted when the output image is determined to be usable. We otherwise then attempt a variety of additional thresholding techniques:

- Otsu's thresholding, named after Nobuyuki Otsu, which is a widely used automatic thresholding technique for image segmentation. The primary goal of Otsu's method is to find an optimal threshold that minimizes the intra-class variance while maximizing the inter-class variance of pixel intensities in a gray scale image. This threshold effectively separates the image into two classes, typically foreground and background, resulting in a binary image. The algorithm calculates the histogram of pixel intensities in the gray scale image and then iterates through all possible threshold values. For each threshold, it computes the intra-class variance, representing

the spread of intensities within each class, and the inter-class variance, representing the difference between the mean intensities of the two classes. The threshold that maximizes the ratio of inter-class variance to intra-class variance is chosen as the optimal threshold. Otsu's method is particularly effective in scenarios where there are distinct intensity peaks corresponding to different image regions. It is robust in handling images with bimodal intensity distributions. This automated thresholding technique is widely employed in various image processing applications, including medical image analysis, document processing, and computer vision tasks, offering a data-driven approach for effective image segmentation.

- Li's thresholding which is an automatic thresholding method used for image segmentation, particularly in scenarios where Otsu's method may not perform optimally. Developed by Cheng-Chang Li, this technique aims to find a threshold that minimizes the cross-entropy between the original grayscale image and the resulting binary image. Unlike Otsu's method, Li's thresholding is suitable for images with uneven illumination or non-uniform background. Li's method involves computing the histogram of pixel intensities and iteratively determining the threshold that minimizes the cross-entropy. Cross-entropy is a measure of the dissimilarity between two probability distributions, and in this context, it represents the dissimilarity between the grayscale image and the binary image based on the chosen threshold. One of the advantages of Li's thresholding is its adaptability to images with varying lighting conditions, making it suitable for a broader range of applications. This method has found use in fields such as medical image analysis, document processing, and industrial quality control. As with any thresholding technique, it is essential to evaluate its performance on specific image characteristics and adjust parameters accordingly for optimal results in diverse imaging scenarios.
- Mean thresholding which is a simple yet effective technique for image segmentation, particularly in cases where the image exhibits a relatively uniform background. This method calculates a threshold based on the mean intensity of the pixel values in the grayscale image. The idea is to classify pixels as foreground or background depending on whether their intensity is above or below the computed mean threshold. The process involves calculating the mean intensity of all pixels in the image and using this value as the threshold. Pixels with intensities greater than the mean are assigned to one class (often considered foreground), while pixels with intensities less than or equal to the mean are assigned to the other class (typically background). This straightforward approach makes mean thresholding computationally efficient and easy to implement. However, mean thresholding may be sensitive to variations in image background and lighting conditions. It may not perform well in cases where the image has a non-uniform background or contains significant noise. As a result, mean thresholding is often most effective in situations where the image exhibits consistent illumination and a clear intensity distinction between foreground and background. Careful consideration of image characteristics is essential when choosing an appropriate thresholding method for optimal segmentation results.

- Yen's thresholding method, proposed by Chin Yen in 1995, which is an automatic image thresholding technique designed to address challenges presented by uneven illumination and varying backgrounds in gray scale images. It aims to find an optimal threshold that maximizes the criterion known as the Yen's entropy. This criterion is based on the information entropy, a measure of uncertainty or disorder in a probability distribution. The Yen thresholding algorithm computes the histogram of pixel intensities and then iteratively evaluates the entropy for all possible threshold values. The threshold that maximizes the Yen's entropy criterion is selected as the optimal threshold for segmenting the image into two classes. Yen's method is particularly effective in scenarios where Otsu's method may struggle, such as images with uneven illumination or complex backgrounds. By considering the information entropy, Yen's thresholding provides a robust solution for images with diverse intensity distributions. This technique has found applications in various fields, including medical image analysis, document processing, and object recognition. Its adaptability to different image characteristics and its ability to handle challenging lighting conditions make Yen's thresholding a valuable tool in automated image segmentation tasks, offering improved performance in situations where traditional methods may fall short.
- Adaptive thresholding, which is a versatile image segmentation technique that addresses challenges posed by variations in illumination across an image. Unlike global thresholding methods, which use a single threshold for the entire image, adaptive thresholding dynamically adjusts the threshold locally based on the pixel values in the vicinity of each image point. The algorithm divides the image into smaller regions or tiles, and a distinct threshold is computed for each region. This enables adaptive thresholding to handle images with uneven lighting or complex backgrounds more effectively. Common methods for adaptive thresholding include mean-based, Gaussian-based, and Sauvola's method, each with its own approach to computing local thresholds. Mean-based adaptive thresholding calculates the threshold for each region by considering the mean intensity of the pixels within that region. Similarly, Gaussian-based methods use the weighted average of pixel intensities, giving more significance to the central pixels. Sauvola's method takes into account both the mean and the standard deviation of pixel intensities to adaptively compute thresholds. Adaptive thresholding is particularly useful in applications such as document processing, character recognition, and medical imaging, where lighting conditions may vary across an image. By adapting to local characteristics, this technique enhances the accuracy of segmentation in diverse scenarios, offering a more robust solution to challenges presented by complex image structures and lighting variations.
- Niblack thresholding is an adaptive thresholding technique designed to address challenges in image segmentation posed by variations in illumination and noise. Proposed by Wayne Niblack in 1986, this method computes local thresholds for each pixel based on the mean and standard deviation of pixel intensities within a local neighborhood or window. The algorithm divides the image into non-overlapping windows and calculates a threshold for each window. Pixels with

intensities higher than the local mean plus a user-defined parameter (typically a multiple of the standard deviation) are classified as foreground, while pixels below this threshold are classified as background. This adaptive approach makes Niblack thresholding well-suited for images with uneven illumination or varying background conditions. One advantage of Niblack thresholding is its sensitivity to local image characteristics, enabling it to handle variations in lighting and noise. However, it may be sensitive to the choice of parameters and may not perform optimally in all scenarios. Despite this, Niblack thresholding has found applications in document image analysis, where text may be present against varying background intensities, and in scenarios where local adaptability is crucial for accurate image segmentation. Experimentation and parameter tuning are often necessary to optimize its performance for specific imaging conditions.

Our SDK (class `X9ImageThresholding`) and `X9Utilities` products utilize all of these thresholding techniques to achieve best possible results. We have done a lot of research and subsequent work to implement a very good solution for these issues. We are interested in your feedback as to how our current solution works and can be further improved.

MICR Line Format and Standards

Magnetic Ink Character Recognition (MICR) technology was adopted in the US in the late 1950's as a standard mechanism to electronically and accurately read check information using the technology that existed at that time. The encoded information identifies the financial institution that issued the check and the account that is associated with the transaction. Numerous standards are defined which identify where the information must be printed and how it must be formatted.

The MICR line is printed using magnetic ink or toner, which is read using a MICR reader. Use of magnetic ink allowed the data to be read even when it was written over or otherwise obscured by subsequent information that was printed on the physical check.

Newer technologies allow information to be more easily captured using Optical Character Recognition (OCR). Many devices today will do a combination of MICR and OCR reads which then compare the results for improved quality.

MICR Line Standards

There are standards that govern the placement and format of some fields of information printed in the MICR data of a check. The fact that standards do not cover the location or meaning of all the information contained in the MICR data of a check presents a problem for parsing operations. The process of inspecting the MICR data information and separating particular fields of information can be done by the MICR reader or host application. In any case, a set of rules must be developed to separate the various information fields. This will only work on checks whose MICR data format follows industry conventions. Once the fields are separated, the information is reformatted for processing by an on-line check processing and clearing service.

The MICR line contains 65 positions, numbered from right to left and grouped into four fields:

- Auxiliary On-U's
- Transit
- On-U's
- Amount

All checks have at least three of the fields (amount, On-U's, and transit number). Commercial checks have an additional field on the left of the check, called the auxiliary On-U's field. Some checks also have an external processing code (EPC) digit, located between the transit and auxiliary On-U's fields. The amount and transit fields have a standardized content, while the contents of the On-U's fields can vary to meet the individual bank's requirements.

MICR Line Parsing

The X9Ware SDK includes class X9MicrLineParser which includes our standard logic which will parse captured MICR line data into their component fields. This class requires that you provide the characters your MICR line symbols, since they can vary based on your scanner. The SDK also includes class X9MicrParserFactory which can be used to allocate new X9MicrLineParser instances using the MICR symbols that are present in an externally defined x9header XML file.

MICR Line Characters





E-13B

There are two types of characters in the E-13B font: numbers and symbols.

The ten numeric characters of the font are 0-9:

0 1 2 3 4 5 6 7 8 9

The four symbols used to control the interpretation of the MICR line include:

 Transit Symbol
 Dash Symbol
 On-Us Symbol
 Amount Symbol

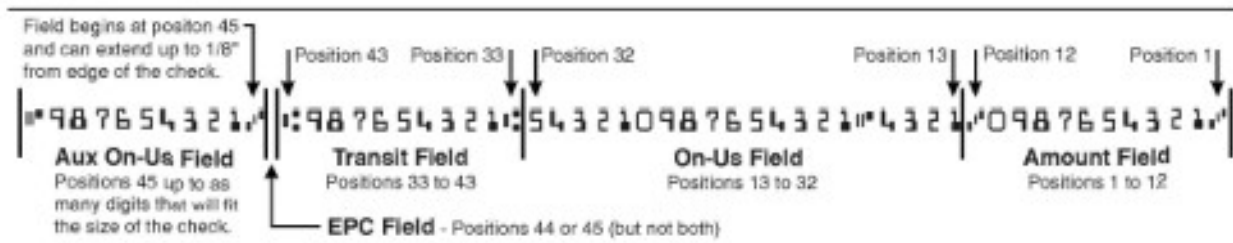
MICR Line Fields

MICR line fields (from right to left on the check) are as follows:

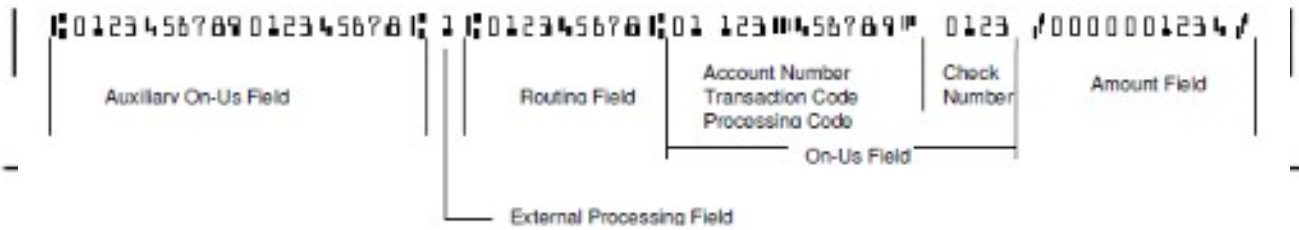
Field #	Field Name	MICR Positions	Description
1	Amount	1 – 12	Amount with leading and trailing E13B amount symbols. This field is typically not encoded in the image environment.
2	On-Us	13– 32	On-Us identifies the customer account and may contain other information such as the check serial number, transaction code, or both. The last position of this field is usually followed by a

Field #	Field Name	MICR Positions	Description
			blank in position 32.
3	Transit	33 – 43	Nine-character routing number with leading and trailing E13B transit symbols. The transit field identifies the payor financial institution. On a check having four fields, the transit field is second from the left. However, shorter personal checks will not have an Auxiliary On-Uss field, and in that situation the transit field is the left-most field of the three fields that are present. US (FRB) routing numbers will typically be a nine-digit number where the last digit is calculated using a MOD10 algorithm. You will also see US routings formatted as xxxx-xxxx (with an embedded dash). You may also encounter Canadian items which are formatted as xxxxx-xxx.
4	EPC	44	The external processing code (EPC) is an optional field that is encoded between the transit and auxiliary On-Uss fields in position 44. When present, this field indicates that the document is eligible for special processing.
5	Auxiliary On-Uss	45-65	The auxiliary On-Uss field is an optional field which is typically used by the payor bank for business check serial numbers or other internal information. When present, it is left-most on the check in MICR line positions 45 through 65. (actual number of potential characters is dependent on the physical width of the item). Aux OnUss is not present on personal checks because of physical size of those items.

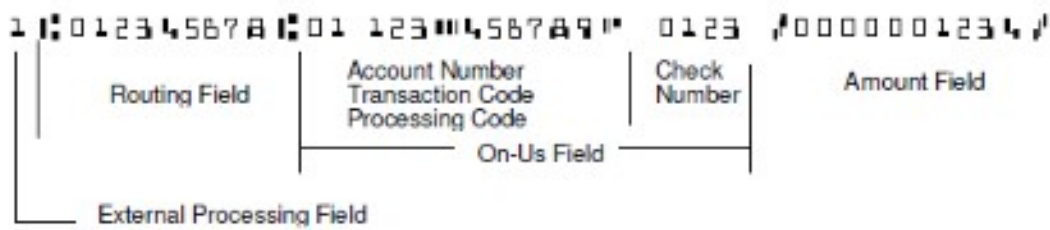
MICR Line Layout



Typical Information Fields on a Business Check



Typical Information Fields on a 6-inch Personal Check



MICR Line RegEx

RegEx matches are usually “greedy” so they will match as many characters as possible. This means using a wildcard character can be used to match everything. For example,

- A* would match all of the A’s in AAAAAAAAAAAAAAAAAAAAAAB,
- A+ would also match them, A would match the first one,
- A{10} would match the first 10,
- And so on.

Commonly used RegEx expressions:

(?<=) - this looks for a match to whatever terms are after the = but does not return it, when put in front of a search it has to match this first. Effectively acts as a left boundary.

(?=) - this looks for a match to whatever terms are after the = but does not return it, when put after of a search it has to match this last. Effectively acts as a right boundary.

\d = any digit.

[A] = match any A.

[ABC] = match any A, B, or C character.

[0-9] = match any digit from 0-9.

[0-9]+ = match all digits in a row, minimum 1.

[0-9]* = match any number of digits in a row (including none).

^ = start of a line.

\$ = end of a line.

\ = used as an escape character, e.g. \\ matches the \ character.

? = after a character or ()? Makes that term optional (greedy means it will include it if it there).

() = group terms and also creates the bracket contents as a variable (variable is referenced as a number based on the order of the opening (e.g. first () is 1, and so on, can be inside brackets themselves.

\1 \$1 = depends on implementation but can be used to reference the value of the corresponding term in brackets.

Based on the above:

Field	RegEx	Regex Notes
Amount	(?<=B)\d+	Matches the part of a string preceded by B that consists of only numbers - it will get all the numbers and stop when it reaches anything not a number.
On-Us	(?<=A)[0-9DC]+(?=C B \$)	Matches the part of a string preceded by A, that contains numbers, C, or D and ends with B, C or the end of the line.
Transit	[0-9D]+(?=A)	Matches the part of a string that precedes A and has numbers or D.
EPC	(?<=^ B)[0-9](?=A)	Matches a single number that is preceded by B or the start of the line, and is followed by A.
Aux On-Us	[\dD]+(?=CA)	Matches the part of a string that consists of digits and D, and is followed by CA.

Further RegEx Reading

<https://www.regular-expressions.info/>

<https://regexr.com/>

Appendix: HeaderXml

Many financial institutions and third party processors have implemented their own x9.37 requirements and variants that are based (to varying degrees) on the x9.37 file standards. The process of generating x9 files generically in the formats required for these processors becomes a complex task given the numerous options and settings that are required.

X9Ware has addressed this need through our HeaderXml class which is implemented within the SDK and leveraged by our X9Utilities product. HeaderXml define parameter values which control the generation of an x9 file. HeaderXml specifically defines the various values that can be populated in the file header, cash letter header, bundle header, and item records.

HeaderXml values are populated from an external XML file. Our long term goal is to provide the options needed to create x9.37 files for virtually all financial institutions and third party processors that use the x9.37 standard. We are largely met that goal today, since we are not aware of any banks with options that we cannot support. This includes all options needed to populate header and trailer records, various credit formats, various credit locations, and a wide variety of parameters that control the values associated with item and image definitions. In alignment with our support goal, be aware that this definition will change from release to release as we continue to improve upon this process and thus expand the parameters. Although we will always make every attempt to retain compatibility with current implementations, you should also design your application and support processes in a manner where you can adapt to ongoing change.

When creating a new HeaderXml file, you should begin with the sample x9headers.xml as included in our software installation. You can then review the field names within this xml file and refer to the user guide for their specific purpose. If you are upgrading from a previous release, you can copy and paste the values from your previous definition. Do this carefully since there is the potential that fields have been moved within the parameters and that field names have been changed to improve clarity.

Editing HeaderXml

Our X9Validator and X9Assist desktop products include the HeaderXml937 Editor, which is tools that can be used to edit, validate, and save HeaderXml definitions. This is the easiest way to create and maintain your HeaderXml files. We highly suggest that all X9Utilities also have X9Validator, since it is the best tool in the industry to validate the x9.37 files that are created by X9Utilities.

XML documents have a hierarchical structure and can conceptually be interpreted as a tree structure, called an XML tree. All XML documents contain a root element (one that is the parent of all other elements). The XML document then contains a series of elements, where each element can itself contain sub-elements, text and attributes.

During the editing process, it is extremely important that the proper tools and file validations be utilized to ensure that editing does not result in an invalid XML file structure. Without this, it is far too easy to save a file that has unmatched XML control tags. When this happens, the XML file cannot be successfully parsed and will ultimately result in an application “abort” when you attempt to use the file.

There are many XML editors that are available in the marketplace today that address these issues. Many environments have chosen and implemented such tools, and you can certainly use your standard tools when available. If you do not have an XML editor immediately available to you, we recommend that you consider one of the following:

- Our X9Validator/X9Assist desktop products include the HeaderXml937 Editor that is targeted specifically for viewing, creating, and modifying these HeaderXml files. Our editor understands our XML format and makes it very easy to manipulate these files. The HeaderXml937 Editor is a standard feature of X9Validator/X9Assist, and was added as part of our R4.05 release. We highly suggest that you consider use of this tool. The functionality provided by the HeaderXml937 Editor is described as the last topic in this user guide.
- Another popular tool is NotePad++ with the XML Tools plugin. This combination provides immediate feedback on XML syntax and will not let you save an XML file with an invalid hierarchical structure. NotePad++ with the XML Tools plugin will ensure that you have matching tags within your XML document, and that using NotePad++ without the XML Tools plugin is a regression back to a simple text editor. However, even with the plugin, NotePad++ cannot validate that the tags themselves are correct, as can be done by X9Validator/X9Assist.
- Another commonly used tool is the XML Notepad editor from Microsoft, which provides a simple intuitive user interface for browsing and editing XML documents. It has similar + / - as using NotePad++.
- Finally, you can revert to using a simple text editor such as Microsoft NotePad. However, doing so forces you to assume complete responsibility for the XML document structure.

HeaderXml as Written to the Log

X9Utilities will write all current HeaderXml settings to the in the system log each time that they are used by the “-write” function. You can use the system log for several determinations.

- You can determine the value that has been assigned to all HeaderXml fields.
- You can review the list of all possible fields which are available. This is extremely useful, since it allows you to see any new parameters that have been added in recent releases.
- You can identify new HeaderXml fields which are available but are not present in the provided xml definition.

The following shows a field value setting when the field is defined in the xml definition:

```
2015-12-03 15:19:50.549 [INFO] document(HeaderXml) fieldName(x9fileSpecification) value(x9.37)
(com.x9ware.dom.X9Dom.getFieldsUsingReflection:624)
```

The following shows a field which is assigned a default value when not defined in the xml:

```
2015-12-03 15:19:50.581 [INFO] document(HeaderXml)fieldName(itemAddendumCount) default(0)
(com.x9ware.dom.X9Dom.getFieldsUsingReflection:624)
```

X9 File Structure

The created x9 file will consist of a single cash letter that is wrapped by a file header and file control trailer. No bundles will exist when a file does not have any items. Bundles are automatically created from the provided items. Individual bundle size is automatically limited by the identified bundle size count.

Inclusion of Credits in Trailer Totals

There are unfortunately no industry wide standards as to how credits are included in bundle, cash letter, and file control trailers. Specific actions to include credits in trailers are thus dependent upon the current x9 file specification and variant being used.

The SDK must be able to both create and validate totals. For convenience, the flags which indicate how credits impact trailers are defined in the x9 headers XML and then replicated in our x9 rules. Setting either of these will result in credits being included in your trailer counts and amounts.

The SDK first interrogates the values defined by the x9.100-187-2013 specification which are optionally present in the bundle, cash letter, and file control trailers to indicate if those specific record types are to include credit counts. A value of “1” indicates that credits add to counts and amounts, while a value of “0” indicates that credits do not add to counts and amounts. These values are take priority over all other settings when present. Note that this standard is flexible but has several oddities. First is that it creates the unusual situation where you might add credits to bundles and not to cash letters. Second is that it does not support the situation where credits add to counts but not amounts.

The SDK otherwise uses our x9 headers XML and x9 rules definitions to determine when and how credits impact the trailer records. There are separate flags to indicate if credits should be added to either counts and/or amounts. Turning a flag on will roll credits through the various levels (bundle, cash letter, and file control) for consistent balancing. There is no current capability to update one level and then forcibly omit in others, since our design is to roll these accumulators through these hierarchies. The SDK does support the ability to include credits in counts but to then exclude them from total amount, which is used by some x9 variants.

HeaderXml Fields defined within the <info> group

The <info> group is used for change management documentation. These fields will be listed to the log in support of problem determination but are otherwise not used.

XML Group	XML Field Name	Populated Into	Notes
<info>	accountName	Credit 52.19	Primarily used for documentation, but also inserted into the credit image when <creditImageDrawFront> is true. When using credit profiles, the account name

XML Group	XML Field Name	Populated Into	Notes
			should be redirected to the profile and this field can instead be set to something generic like “various”.
<info>	bankName	Credit 52.19	Primarily used for documentation, but also inserted into the credit image when <creditImageDrawFront> is true.
<info>	author	N/A	Used for documentation only.
<info>	dateWritten	N/A	Used for documentation only.
<info>	dateModified	N/A	Used for documentation only.
<info>	comments	N/A	Used for documentation only.

HeaderXml Fields defined within the <fields> group

The HeaderXml values that can be populated are defined below. This definition was substantially changed with the R3.03 release so it must be reviewed closely.

XML Group	XML Field Name	Populated Into	Notes
<fields>	x9fileSpecification	N/A	Identifies the x9 file specification to be created. Default is "x9.37".
<fields>	businessDate	10.05	Numeric; default is current YYYYMMDD.
<fields>	createDate	01.06, 10.06	Numeric; default is current YYYYMMDD.
<fields>	createTime	01.07, 10.07	Numeric and “0000” through “2359”; default is current HHMM when omitted. This value can also be provided as an offset to the current time. For example, a value of “+3” will add three hours to the current system time and that a value of “-2”

XML Group	XML Field Name	Populated Into	Notes
			will subtract three hours. Note that providing a current time offset may also update the current date.
<fields>	batchProfile	N/A	Batch profile is used to assign a static profile name which will default to “”. Batch profiles are an advanced function where the profile name would typically be specified on each incoming item CSV row. In those situations, the items are reordered and batched by profile, and the profile name can be used to redirect certain headerXml values to an external properties file. However, it is also possible to statically assign a single batch profile name to the headerXml file. When doing this, you can still redirect certain field assignments to an external properties file. An example of usage would be <batchProfile>customer.properties</batchProfile>.
<fields>	fileStandardLevel	01.02	Default is “03”.
<fields>	fileMode	01.03	Default is “T”.
<fields>	fileOriginationRouting	01.05	
<fields>	fileOriginationName	01.10	
<fields>	fileDestinationRouting	01.04	
<fields>	fileDestinationName	01.09	
<fields>	fileIdModifier	01.11	If the fileIdModifier value is provided as one character, then it represents the specific value to be assigned. It otherwise is the same of a fileIdModifier xml file will be referenced and used to assign a rolling fileIdModified within the current calendar date. This external file reference can be provided on an absolute (fully qualified) or relative basis (location would be within the

XML Group	XML Field Name	Populated Into	Notes
			same folder where this headerXml definition appears).
<fields>	fileResendIndicator	01.08	
<fields>	fileUcdIndicator	01.14	
<fields>	fileCountryCode	01.12	
<fields>	fileUserField	01.13	
<fields>	cashLetterEceInstitutionRouting	10.04	
<fields>	cashLetterDestinationRouting	10.03	
<fields>	cashLetterIdentifier	10.10	<p>Default (when this field is omitted) is to create as “hmmsss” which satisfies the common requirement that the cash letter identifier be unique for a given day. There are several other alternatives:</p> <ul style="list-style-type: none"> • “xxxxxxx” (up to 8 character string) which is directly assigned to all cash letters. • “sequential” which assigns an incremented cash letter identifier beginning with “00000001”. • “creditISN” which assigns the rightmost 10 characters of the credit item sequence number to the bundle identifier. This feature requires that creditBeginsNewBundle is true. • “creditSerial” which assigns the rightmost 10 characters of the credit AuxOnUs serial number to the bundle identifier. This feature requires that creditBeginsNewBundle is true. • “%xxxx” (up to a 4 character user string) which inserts the variable length cash letter

XML Group	XML Field Name	Populated Into	Notes
			<p>number at the beginning making it unique. For example, a value of “%BI” assigns a value of “1BI” to the first cash letter, while “%” would simply assign a value of “1”.</p> <ul style="list-style-type: none"> • “xxxx%” (up to a 4 character user string) which inserts the variable length cash letter number at the end making it unique. For example, a value of “BI%” assigns a value of “BI1” to the first cash letter. • “#xxxx” (up to a 4 character user string) which inserts the current four character cash letter number at the beginning making it unique. For example, a value of “#BI” assigns a value of “0001BI” to the first cash letter, while “#” would simply assign a value of “0001”. • “xxxx#” (up to a 4 character user string) which inserts the current four character cash letter number at the end making it unique. For example, a value of “BI#” assigns a value of “BI0001” to the first cash letter.
<fields>	cashLetterContactName	10.11	
<fields>	cashLetterContactPhone	10.12	Numeric
<fields>	cashLetterReturnsIndicator	10.14	
<fields>	cashLetterRecordTypeIndicator	10.08	
<fields>	cashLetterDocumentationTypeIndicator	10.09	
<fields>	cashLetterCollectionTypeIndicator	10.02	
<fields>	cashLetterFedWorkType	10.13	

XML Group	XML Field Name	Populated Into	Notes
<fields>	cashLetterUserField	10.15	
<fields>	bundleItemCount	N/A	Default is 300.
<fields>	bundleEceInstitutionRouting	20.4	Defaulted from the current cash letter header when omitted.
<fields>	bundleDestinationRouting	20.3	Defaulted from the current cash letter header when omitted.
<fields>	bundleIdentifier	20.07	<p>Default (when this field is omitted) is to create as YYMMDDHHMM which is unique for a given calendar day for the current destination. This setting satisfies the common requirement that the combination of bundle identifier and bundle sequence number are unique within a single x9 file. There are several other alternatives:</p> <ul style="list-style-type: none"> • “xxxxxxxx” (up to 10 character string) which is directly assigned to all bundles. • “sequential” which assigns an incremented bundle identifier beginning with “0000000001”. • “%xxxxxx” (up to a 6 character user string) which inserts the variable length bundle number at the beginning making it unique. For example, a value of “%BI” assigns a value of “1BI” to the first bundle, while “%” would simply assign a value of “1”. • “xxxxxx%” (up to a 6 character user string) which inserts the variable length bundle number at the end making it unique. For example, a value of “BI%” assigns a value of “BI1” to the

XML Group	XML Field Name	Populated Into	Notes
			<p>first bundle.</p> <ul style="list-style-type: none"> • “#xxxxxx” (up to a 6 character user string) which inserts the current four character bundle number at the beginning making it unique. For example, a value of “#BI” assigns a value of “0001BI” to the first bundle, while “#” would simply assign a value of “0001”. • “xxxxxx#” (up to a 6 character user string) which inserts the current four character bundle number at the end making it unique. For example, a value of “BI#” assigns a value of “BI0001” to the first bundle. • “creditISN” which assigns the rightmost 10 characters of the credit item sequence number to the bundle identifier. This feature requires that creditBeginsNewBundle is true.
<fields>	bundleCycleNumber	20.09, 52.04	This field is optional, but when provided, it will be consistently populated into the bundle header record (20.09) and the image view data record (52.04).
<fields>	bundleReturnsRouting	20.10	
<fields>	bundleUserField	20.11	
<fields>	bundleReservedField	20.12	
<fields>	The fields from this point forward are included in the XML parameters file and are used exclusively by the X9Writer interface provided via the SDK. They can then be utilized by X9Utilities when the HeaderXml file is used for your writer		

XML Group	XML Field Name	Populated Into	Notes
	parameters.		
<fields>	bundleSequenceNumberAlpha	20.08	Populates the bundle sequence number on an alphanumeric basis when set to true. For example, the first bundle will be assigned a value of “1” when this parameter is enabled. The default is false where the first bundle will instead be assigned a value of “0001”.
<fields>	trailerInstitutionName	90.06	
<fields>	trailerSettlementDate	90.07	Default is business date when not provided; a value of “none” will cause the settlement date to be set to spaces.
<fields>	trailerContactName	99.06	
<fields>	trailerContactPhone	99.07	Numeric
<fields>	trailerCreditTotalIndicator	70.07, 90.08, 99.08	When using the x9.100-187-2013 standard, specifies the credit total indicator value that should be set in trailer records. This field has values of “1” (accumulated credits into trailers) or “0” (do not accumulate credits into trailers).
<fields>	trailerPopulateMicrValidAmount	70.04	Default is “true”. Indicates if the accumulated MICR valid amount should be populated into the bundle trailer record.
<fields>	trailerPopulateImageCount	70.05	Default is “true”. Indicates if the accumulated image count should be populated into the bundle trailer record.
<fields>	creditFormat <ul style="list-style-type: none"> • “metavante” – an industry standard type 61 credit with 13 fields that is defined in x9rules as format 61-001. • “dstu” – an industry standard type 		Identifies the credit record type and format to be used to create the credit per the selected x9 configuration rules. This field can be populated in one of several manners. First, on a logical basis using a record level description that is set within x9

XML Group	XML Field Name	Populated Into	Notes
	<p>61 credit with 12 fields that is defined in x9rules as format 61-002.</p> <ul style="list-style-type: none"> • “x9.100-180” – an industry standard type 61 credit which is 84 characters long that is defined in x9rules as format 61-003. • “wellsfargo” – an industry standard type 61 credit with 11 fields that is defined in x9rules as format 61-004. • “t25” – an alternative that uses a type 25 check detail record to represent the credit. The item must be identified as a credit in some manner, typically using an appended transaction code in MICR OnUs, but possibly also using a dedicated credit routing. • “t10” – an alternative that batches each deposit within a dedicated cash letter. Each deposit account must be identified in some manner, typically using either the cash letter ECE origination routing, the contact name, or the contact phone number. When using this format, the credit record location must be set to “none”. 		<p>rules (eg, “metavante”) or on an absolute basis using the record format (eg, “61-001”).</p> <p>Various options (from those that follow) must be used to further configure the constructed credit.</p> <p>“creditInsertedAutomatically” must be enabled to activate this feature.</p> <p>“creditLocation” must be assigned to define where the generated credit will be inserted into the file.</p> <p>The default is that all credits will begin in a newly created bundle.</p> <p>Images can be either dynamically drawn or provided from external image files.</p> <p>Primary and secondary endorsements can be created and attached to the credit.</p> <p>The impact that this credit will have against the trailer records can be defined. This directs how the credit will impact count and amount totals that are present in the bundle, cash letter and trailer records.</p>
<fields>	<p>creditRecordLocation</p> <p>Supported values are:</p> <ul style="list-style-type: none"> • none => credit is not to be inserted • a01 => after the file header • a10 => after the cash letter header • a20 => after the bundle header • b70 => before the last bundle trailer for all items within the current deposit (transaction) • a90 => after the cash letter trailer 		<p>Identifies the location where the credit should be inserted into the created x9 file.</p> <p>A value of “none” indicates that the credit is not to be inserted, which is a convenient way to allow a defined credit to be turned on or off during initial testing.</p> <p>The most commonly used setting is “a20” which will insert the credit after the first bundle header record.</p> <p>A value of “none” must be used when</p>

XML Group	XML Field Name	Populated Into	Notes
			a credit format of “t10” has been assigned, since there is no actual credit to be inserted.
<fields>	<p data-bbox="300 434 847 594">creditInsertedAutomatically This facility is used to insert one or more deposit tickets into the created file when required by the receiving bank.</p> <p data-bbox="300 644 847 825">Content of the deposit ticket must be defined using the other credit xml fields below (creditPayorBankRouting, creditMicOnUs, creditMicAuxOnUs, creditItemSequenceNumber, etc).</p> <p data-bbox="300 875 847 1056">The deposit ticket can optionally contain attached proxy images which are created using external tiff images which are identified by creditImageProxyFront and creditImageProxyBack.</p>		Default is “false”. Indicates that a credit should be automatically generated using an amount which is calculated as the sum of all debits (checks) which are present in the current file.
<fields>	<p data-bbox="300 1096 847 1318">creditStructure Provides further direction regarding the creation of individual credits. This parameter is applicable only when creditInsertedAutomatically has been enabled.</p> <p data-bbox="300 1327 847 1465">Credit structure allows the checks with the deposit to be grouped in specific ways, subject to customer or financial institution requirements.</p> <p data-bbox="300 1474 847 1850">When using “bundledCredits”, it is important to format your csv file such that an item record (t25, t31, 25, 31, etc) appears before other csv lines for this same item. For example, the paidStamp must be after a t25 line (and the paidStamp must be before the image line). This is important because the csv lines will be grouped and reordered when constructing the deposits, so the item record itself must always be first.</p>		<p data-bbox="1015 1096 1507 1255">“multiItem” creates a credit which is offset by multiple checks. This is the default value and represents standard processing.</p> <p data-bbox="1015 1264 1507 1360">“singleItem” creates single item deposits (every check is created within its own deposit).</p> <p data-bbox="1015 1369 1507 1759">“bundledCredits” creates bundles that will each contain their own credit. This option is applicable only when the financial institution requires that each bundle contains a credit (deposit ticket). Several other parameters work in conjunction with this option. You must enable creditBeginsNewBundle and then set bundleItemCount to the maximum number of checks that should be attached to each credit.</p>

XML Group	XML Field Name	Populated Into	Notes
<fields>	creditAccountName	Credit 52.19	Account name inserted into the credit image when <creditImageDrawFront> is true, and will override info account name (in the header) when present.
<fields>	creditPayorBankRouting	6x.xx, 25.04	The payor bank routing that is used when a credit is inserted automatically or when the ["credit", amount] format is present on the items csv file.
<fields>	creditMicrOnUs	6x.xx, 25.06	The MICR OnUs that is used when a credit is inserted automatically or when the ["credit", amount] format is present on the items csv file.
<fields>	creditMicrAuxOnUs	6x.xx, 25.02	<p>The MICR AuxOnUs that is used when a credit is inserted automatically or when the ["credit", amount] format is present on the items csv file. A value can be explicitly provided. More commonly, one of our patterns is used to generate the value. The available patterns are as follows:</p> <p>“auto” will assign a 10 digit number as yymmddhhmm.</p> <p>"jjjhhmmnnn" will assign a 10 digit number as jjjhhmmnnn where jjj is the Julian day within the current year and nnn is a sequential number that is incremented for each new credit.</p> <p>“hmmssnnn” will assign a 10 digit number where nnnn is a sequential number that is incremented for each new credit.</p> <p>“debitSequenceNumber” will assign up to a 10 digit number that is taken from the first item in the attached deposit. This will be the right-most 10 digits of that sequence number (which can be up to 15 digits). Using this value can facilitate correlation of</p>

XML Group	XML Field Name	Populated Into	Notes
			the credit back to the attached items.
<fields>	creditItemSequenceNumber	6x.xx, 25.08	<p>The item sequence number that is used when a credit is inserted automatically or when the generic “credit” format is used. A value can be explicitly provided. More commonly, one of our patterns is used to generate the value. The available patterns are as follows:</p> <p>“auto” will assign a 15 digit number as yymmddhhmmssnnn where nnn is a sequential number that is incremented for each new credit.</p> <p>"yyjjhhmmssnnn" will assign a 15 digit number as yyjjdhmmssnnnn where jjj is the Julian day within the current year and nnnn is a sequential number that is incremented for each new credit.</p> <p>“yyyymmddhhmmss” will assign a 14 digit number as yyyymmddhhmmss.</p>
<fields>	creditRecordUsageIndicator	6x.xx	
<fields>	creditDocumentationTypeIndicator	6x.xx, 25.09	
<fields>	creditTypeOfAccount	6x.xx	
<fields>	creditSourceOfWork	6x.xx	
<fields>	creditWorkType	6x.xx	
<fields>	creditDebitCreditIndicator	6x.xx	
<fields>	creditReturnAcceptanceIndicator	25.10	Used when credits are populated as t25.
<fields>	creditMicrValidIndicator	25.11	Used when credits are populated as t25.
<fields>	creditBofdIndicator	25.12	Used when credits are populated as t25.
<fields>	creditAddendumCount	25.13	Used when credits are populated as t25.
<fields>	creditCorrectionIndicator	25.14	Used when credits are populated as

XML Group	XML Field Name	Populated Into	Notes
			t25.
<fields>	creditArchiveTypeIndicator	25.15	Used when credits are populated as t25.
<fields>	creditBeginsNewBundle		Default is “true”. Indicates that each credit should automatically begin a new bundle.
<fields>	creditImageDrawFront	Front Image	Default is “false”. Indicates that the front image should be automatically drawn as a deposit slip from the credit information.
<fields>	creditImageDrawBack	Back Image	Default is “false”. Indicates that the back image should be inserted as a “blank” image.
<fields>	creditImageDrawMicrLine	Front Image	Default is “false”. Indicates that the micr line should be included in the drawn front image.
<fields>	creditImageTitle	Front Image	Default is “Remote Deposit”. Provides the document title that is included in the drawn front image.
<fields>	creditImageDrawCheckListCount	Back Image	Default is 15. Provides the number of lines to be included within a simulated list of deposited items. Setting this value to zero will eliminate the list completely. This list is provided only as back side image content, to ensure that this image will not fail an IQA too light test (which may happen if the image is totally blank).
<fields>	creditImageProxyFront	Front Image	
<fields>	creditImageProxyBack	Back Image	
<fields>	creditCreateBofd		Default is “false”. Indicates that a BOFD type 26 addendum should be created and attached to the credit.
<fields>	creditCreateSecondaryEndorsement		Default is “false”. Indicates that a secondary type 28 addendum should

XML Group	XML Field Name	Populated Into	Notes
			be created and attached to the credit.
<fields>	creditAddToItemCount	70.02, 90.03, 99.03	Default to “false” since these counts typically only include the debits.
<fields>	creditAddToItemAmount	70.03, 90.04, 99.04	Default to “false” since these counts typically only include the debits.
<fields>	creditAddToImageCount	70.05, 90.05	Default to “true” since these counts typically include all type 52 records (debits and 61/62 credits).
<fields>	itemDocumentationTypeIndicator	25.09	This value will be assigned to all items that are defined using “t25” item rows, since those rows include only basic item information and do not include the various type 25 indicator values.
<fields>	itemReturnAcceptanceIndicator	25.10	Usage is as documented above for field itemDocumentationTypeIndicator.
<fields>	itemMicrValidIndicator	25.11	Usage is as documented above for field itemDocumentationTypeIndicator.
<fields>	itemBofdIndicator	25.12	Usage is as documented above for field itemDocumentationTypeIndicator.
<fields>	itemAddendumCount	25.13, 31.07	Defaults to zero and without override is populated based on the actual addendum count for this item. Usage is as documented above for field itemDocumentationTypeIndicator.
<fields>	itemCorrectionIndicator	25.14	Usage is as documented above for field itemDocumentationTypeIndicator.
<fields>	itemArchiveIndicator	25.15	Usage is as documented above for field itemDocumentationTypeIndicator.
<fields>	itemImageCreatorRouting	50.3	Defines the routing number of the

XML Group	XML Field Name	Populated Into	Notes
			financial institution which has captured the image. This value is normally omitted and allowed to default to the cashLetterEceInstitutionRouting. Usage is as documented above for field itemDocumentationTypeIndicator.
<fields>	bofdAddendumRouting	26.03	Nine digit routing to be assigned when a type 26 addenda is to be created. As an alternative to a routing number, a value string of “blank” can be assigned which will trigger the creation of this addenda with the routing field blank (this would be an unusual requirement).
<fields>	bofdDepositAccountNumber	26.06	Deposit account number, which normally is assigned to debits from the offsetting credit. This field can be used to assign the deposit account number for those files that do not contain credits.
<fields>	bofdDepositBranch	26.07	Deposit branch.
<fields>	bofdPopulateDepositAccountNumber	26.06	Boolean which defaults to “false”. This can be set to “true” to populate the deposit account number from either the offsetting credit (when one is present) or from the above field bofdDepositAccountNumber (when the file does not contain credits).
<fields>	bofdAddendumTruncationIndicator	26.09	
<fields>	bofdAddendumConversionIndicator	26.10	
<fields>	bofdAddendumCorrectionIndicator	26.11	
<fields>	bofdAddendumUserField	26.12	This field can contain a constant user value or a specially formatted credit/debit marker string. The credit/debit marker is used by some x9 variants to identify credits versus debits, since there are often times no

XML Group	XML Field Name	Populated Into	Notes
			other way to accomplish this. For example, an xmlValue of: "CreditDebit=C:D" assigns "C" for credit and "D" for debit; "CreditDebit=:D" assigns "" for credit and "D" for debit; "CreditDebit=C:" assigns "C" for credit and "" for debit; "CreditDebit=CR:DR" assigns "CR" for credit and "DR" for debit; and so forth.
<fields>	secdAddendumRouting	28.03	Nine digit routing to be assigned when a type 28 addenda is to be created. As an alternative to a routing number, a value string of "blank" can be assigned which will trigger the creation of this addenda with the routing field blank (this would be an unusual requirement).
<fields>	secdAddendumPopulateDate	28.04	Boolean which defaults to "false". This can be set to "true" to populate the item date.
<fields>	secdAddendumPopulateSequenceNumber	28.05	Boolean which defaults to "false". This can be set to "true" to populate the item sequence number.
<fields>	secdAddendumTruncationIndicator	28.06	
<fields>	secdAddendumConversionIndicator	28.07	
<fields>	secdAddendumCorrectionIndicator	28.08	
<fields>	secdAddendumUserField	28.10	
<fields>	secdAddendumBankIdentifier	28.11	
<fields>	A second type 28 addendum can be created using the same fields as above using the prefix "secd2" instead of "secd".	28.xx	Second type 28 endorsement record. The populate date and populate sequence number fields are not duplicated; those fields apply to all secondary addenda records.
<fields>	A third type 28 addendum can be created using the same fields as above using the prefix "secd3" instead of "secd".	28.xx	Third type 28 endorsement record. The populate date and populate sequence number fields are not

XML Group	XML Field Name	Populated Into	Notes
			duplicate; those fields apply to all secondary addenda records.
<fields>	imageDetailImageIndicator	50.02	Default is "1".
<fields>	imageDetailFormatIndicator	50.05	Default is "00".
<fields>	imageDetailCompressionAlgorithm	50.06	Default is "00".
<fields>	imageDetailDataSize	50.7	Default is "blank"; can be set to "zero" which results in the value of zero being assigned; can be set to "actual" with results in the actual image size being assigned when available.
<fields>	imageDetailViewDescriptor	50.09	Default is "0".
<fields>	imageDetailDigitalSignatureIndicator	50.10	Default is "0".
<fields>	imageDetailDigitalSignatureMethod	50.11	
<fields>	imageDetailSecurityKeySize	50.12	
<fields>	imageDetailStartOfProtectedData	50.13	
<fields>	imageDetailLengthOfProtectedData	50.14	
<fields>	imageDetailImageRecreateIndicator	50.15	Default is "0".
<fields>	imageDetailUserField	50.16	
<fields>	imageDetailReserved1	50.17	Applies to x9.100-187 (2008 and 2013).
<fields>	imageDetailOverrideIndicator	50.18	Applies to x9.100-187 (2008 and 2013).
<fields>	imageDetailUserField	50.16	
<fields>	imageDataEceInstitutionRouting	52.2	Will default to 10.4 ECE Institution Routing Number when omitted.
<fields>	imageDataSecurityOriginatorName	52.06	
<fields>	imageDataSecurityAuthenticatorName	52.07	
<fields>	imageDataSecurityKeyName	52.08	
<fields>	imageDataClippingOrigin	52.09	Default is "0".
<fields>	imageDataClippingCoordinateH1	52.10	
<fields>	imageDataClippingCoordinateH2	52.11	
<fields>	imageDataClippingCoordinateV1	52.12	

XML Group	XML Field Name	Populated Into	Notes
<fields>	imageDataClippingCoordinateV2	52.13	
<fields>	imageDataPopulateReferenceKey	52.15	Boolean which defaults to “false”. Used by SDK applications that invoke X9Writer directly to indicate when the reference key should be populated with an item level value.
<fields>	imageDataPopulateDigitalSignature	52.17	Boolean which defaults to “false”. Used by SDK applications that invoke X9Writer directly to indicate when the digital signature should be populated with an item level value.
<fields>	ebcdicEnCoding		Boolean which defaults to “true”. Indicates that the output x9 file should be created in the EBCDIC character set. Indicates (when false) that the x9 file should be created in ASCII.
<fields>	fieldZeroInserted		Boolean which defaults to “true”. Indicates that field zero (the four byte binary record length) should be inserted at the beginning of each x9 record.
<fields>	variableFieldDescriptorsPopulateAsNumeric	52.14, 52.16, 52.18, etc.	Boolean which defaults to “false”. Indicates that variable length field descriptors should always be populated on a numeric basis even when they are defined as numeric blank by the current standard. Either format will pass validation but forcing the value to complete numeric may allow a generated x9 file to be more acceptable to receiving processors.
<fields>	micrTransitSymbol		Default is “A” and is not case sensitive; used to parse the MICR line.
<fields>	micrAmountSymbol		Default is “B” and is not case sensitive; used to parse the MICR line.

XML Group	XML Field Name	Populated Into	Notes
<fields>	micrOnUsSymbol		Default is “C” and is not case sensitive; used to parse the MICR line.
<fields>	micrDashSymbol		Default is “D” and is not case sensitive; used to parse the MICR line.

Appendix: X9 Record Types

Type 25 Check Detail Record

The Check Detail Record represents a single check (item) and may appear only within an active bundle. It is typically present in a forward presentment, cash letter which is identified with a Collection Type Indicator of '00', '01' or '02'. Each type 25 record represents a single item. The data in Fields 2 through 7 represent the check MICR line which was captured from the item.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "25".
2	Aux OnUs	C	3	15	NBSM	
3	External Processing Code (EPC)	C	18	1	ANS	
4	Payor Bank Routing	M	19	8	N	First eight digits of the routing as captured from the MICR line.
5	Payor Bank Routing Check Digit	C	27	1	NBSM	Ninth digit of the routing as captured from the MICR line.
6	On Us	C	28	20	NBSM	
7	Amount	M	48	10	N	Item amount.
8	Item Sequence Number	M	58	15	NB	Your internal sequence number assigned to this item as a unique identification.
9	Documentation Type Indicator	C	73	1	AN	Suggested value "G" which is image included with no paper provided.
10	Return Acceptance Indicator	C	74	1	AN	Suggested value "1" which indicates acceptance of preliminary return notifications, returns, and final return notifications.
11	MICR Valid Indicator	C	75	1	N	Suggested value "1" which indicates good MICR read.
12	BOFD Indicator	M	76	1	A	Suggested value "Y" which indicates that the ECE institution is BOFD.
13	Addendum Count	M	77	2	N	Must be set to 00 when there are not addendums for this check detail

Field	Field Name	Usage	Position	Length	Format	Notes
						record.
14	Correction Indicator	C	79	1	N	Suggested value spaces since the field is conditional. 0' No Repair '1' Repaired '2' Repaired without Intervention '3' Repaired with Operator Intervention '4' Undetermined '4' Undetermined
15	Archive Type Indicator	C	80	1	AN	Suggested value spaces since the field is conditional.

Type 26 Check Detail Addendum A Record

The Check Detail Addendum A Record represents the Bank of First Deposit (BOFD) endorsement for this item. Presence of this record type is conditional and is used to document a specific processing entity within the endorsement chain. There is typically only a single type 26 record for a given item, but that requirement is not absolute subject to clearing arrangements. The type 26 endorsement record must always follow its immediately preceding Check Detail Record (Type 25) or another Check Detail Addendum A Record (Type 28). It is one of three addendum type records which are available for use within the Check Detail Record item group.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "26".
2	Check Detail Addendum A Record Number	M	3	1	N	Assigned sequentially beginning with 1.
3	Bank of First Deposit (BOFD) Routing Number	C	4	9	N	
4	Business (Endorsement) Date	C	13	8	N	
5	Item Sequence Number	C	21	15	NB	

Field	Field Name	Usage	Position	Length	Format	Notes
6	Deposit Account Number at BOFD	C	36	18	ANS	
7	Deposit Branch	C	54	5	ANS	
8	Payee Name	C	59	15	ANS	
9	Truncation Indicator	C	74	1	A	Y' Yes this institution truncated the original check 'N' No this institution did not truncate the original check
10	Conversion Indicator	C	75	1	AN	'0' Did not convert physical document '1' Original paper converted to IRD '2' Original paper converted to image '3' IRD converted to another IRD '4' IRD converted to image of IRD '5' Image converted to an IRD '6' Image converted to another image '7' Did not convert image '8' Undetermined
11	Correction Indicator	C	76	1	N	0' No Repair '1' Repaired '2' Repaired without Intervention '3' Repaired with Operator Intervention '4' Undetermined
12	User Field	C	77	1	ANS	
13	Reserved	M	78	3	B	

Type 27 Check Detail Addendum B Record

The Check Detail Addendum B Record is conditional and is typically used to define the location of an image within an image archive. It should only be present only under defined clearing arrangements. The image archive locator record should always its immediately preceding Check Detail Record (Type 25) or a Check Detail Addendum A Record (Type 26) when present. Only one Check Detail Addendum B Record is permitted for a Check Detail Record (Type 25). It is one of three addendum type records which are available for use within the Check Detail Record item group.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "33".

Field	Field Name	Usage	Position	Length	Format	Notes
2	Variable Size Record Indicator	M	3	1	N	0' this is an 80-byte record; Field 2 has a value of 34. '1' Field 5 is variable size.
3	Microfilm Archive Sequence Number	C	4	15	NB	
4	Length of Image Archive Locator	M	19	4	N	Value must be 1 through 999.
5	Image Archive Locator	C	23	34	ANS	
6	Description	C	57	15	ANS	
7	User Field	C	72	4	ANS	
8	Reserved	M	76	5	B	

Type 28 Check Detail Addendum C Record

The Check Detail Addendum C Record represents a subsequent endorsement for this item. Presence of this record type is conditional and is used to document a specific processing entity within the endorsement chain. There may be multiple type 28 records for a given item and they are sequentially numbered beginning at one. The type 28 endorsement record must immediately follow its Check Detail Record (Type 25), Check Detail Addendum A Record (Type 26), or a Check Detail Addendum B Record (Type 27) when present. It is one of three addendum type records which are available for use within the Check Detail Record item group.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "28".
2	Check Detail Addendum C Record Number	M	3	2	N	Assigned sequentially beginning with 1.
3	Bank Routing	C	5	9	N	
4	Endorsement Date	C	14	8	N	
5	Item Sequence Number	C	22	15	NB	
6	Truncation Indicator	C	37	1	A	Y' Yes this institution truncated the original check 'N' No this institution did not truncate the original check

Field	Field Name	Usage	Position	Length	Format	Notes
7	Conversion Indicator	C	38	1	AN	'0' Did not convert physical document '1' Original paper converted to IRD '2' Original paper converted to image '3' IRD converted to another IRD '4' IRD converted to image of IRD '5' Image converted to an IRD '6' Image converted to another image '7' Did not convert image '8' Undetermined
8	Correction Indicator	C	39	1	N	0' No Repair '1' Repaired '2' Repaired without Intervention '3' Repaired with Operator Intervention '4' Undetermined
9	Return Reason	C	40	1	AN	
10	User Field	C	41	15	ANS	
11	Reserved	M	56	15	B	

Type 31 Return Record

The Return Record represents a single check (item) and may appear only within an active bundle. It is typically present in a return cash letter which is identified by a Collection Type Indicator (10.2) set to a value of '03' (Return), '04' (Return Notification), '05' (Preliminary Return Notification), or '06' (Final Return Notification). Each type 31 record represents a single item that often times is being returned as a result of a type 26 forward presentment item. Note that the Auxiliary On-U's field is not present in this record type, due to a lack of space, and is present in the optional type 32 record which follows.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "31".
2	Payor Bank Routing	M	3	8	N	First eight digits of the routing as captured from the MICR line.
3	Payor Bank Routing Check Digit	C	11	1	NBSM	Ninth digit of the routing as captured from the MICR line.
4	On Us	C	12	20	NBSM	
5	Item Amount	M	32	10	N	

Field	Field Name	Usage	Position	Length	Format	Notes
6	Return Reason	M	42	1	AN	'A' NSF - Not Sufficient Funds 'B' UCF - Uncollected Funds Hold 'C' Stop Payment 'D' Closed Account 'E' UTLA - Unable to Locate Account 'F' Frozen/Blocked Account 'G' Stale Dated 'H' Post Dated 'I' Endorsement Missing 'J' Endorsement Irregular 'K' Signature(s) Missing 'L' Signature(s) Irregular 'M' Non-Cash Item (Non-Negotiable) 'N' Altered/Fictitious Item 'O' Unable to Process (e.g. Mutilated Item) 'P' Item Exceed Dollar Limit 'Q' Not Authorized 'R' Branch/Account Sold (Wrong Bank) 'S' Refer to Maker 'T' Stop Payment Suspect 'U' Unusable Image (Image could not be used for required business purpose) 'V' Image fails security check 'W' Cannot Determine Amount
7	Return Record Addendum Count	M	43	2	N	
8	Return Documentation Type Indicator	C	45	1	AN	'A' No image provided, paper provided separately 'B' No image provided, paper provided separately, image upon request 'C' Image provided separately, no paper provided 'D' Image provided separately, no paper provided, image upon request 'E' Image and paper provided separately 'F' image and paper provided separately, image upon request 'G' Image included, no paper provided 'H' Image included, no paper provided,

Field	Field Name	Usage	Position	Length	Format	Notes
						image upon request 'I' Image included, paper provided separately 'J' Image included, paper provided separately, image upon request 'K' No image provided, no paper provided 'L' No image provided, no paper provided, image upon request 'M' No image provided, Electronic Check provided separately
9	Forward Bundle Date	C	46	8	N	
10	Item Sequence Number	C	54	15	NB	
11	External Processing Code	C	69	1	ANS	
12	Return Notification Indicator	C	70	1	N	'1' Preliminary notification '2' Final notification
13	Return Archive Type Indicator	C	71	1	AN	'A' Microfilm 'B' Image 'C' Paper 'D' Microfilm and image 'E' Microfilm and paper 'F' Image and paper 'G' Microfilm, image and paper 'H' Electronic Check Instrument 'I' None
14	Reserved	M	72	9	B	

Type 32 Return Addendum A Record

The Return Addendum A Record represents the Bank of First Deposit (BOFD) endorsement for this item. Its presence is conditional. There is typically only a single type 31 record for a given item, but that requirement is not absolute subject to clearing arrangements. The type 32 endorsement record must always follow its immediately preceding Return Record (Type 31) or another Return Addendum A

Record (Type 32). It is one of four addendum type records which are available for use with the Return Record item group.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "32".
2	Return Addendum A Record Number	M	3	1	N	Assigned sequentially beginning with 1.
3	Bank of First Deposit (BOFD) Routing Number	C	4	9	N	
4	Business (Endorsement) Date	C	13	8	N	
5	Item Sequence Number	C	21	15	NB	
6	Deposit Account Number at BOFD	C	36	18	ANS	
7	Deposit Branch	C	54	5	ANS	
8	Payee Name	C	59	15	ANS	
9	Truncation Indicator	C	74	1	A	Y' Yes this institution truncated the original check 'N' No this institution did not truncate the original check
10	Conversion Indicator	C	75	1	AN	'0' Did not convert physical document '1' Original paper converted to IRD '2' Original paper converted to image '3' IRD converted to another IRD '4' IRD converted to image of IRD '5' Image converted to an IRD '6' Image converted to another image '7' Did not convert image '8' Undetermined
11	Correction Indicator	C	76	1	N	0' No Repair '1' Repaired '2' Repaired without Intervention '3' Repaired with Operator Intervention '4' Undetermined
12	User Field	C	77	1	ANS	

Field	Field Name	Usage	Position	Length	Format	Notes
13	Reserved	M	78	3	B	

Type 33 Return Addendum B Record

The Return Addendum B Record is conditional and should be present unless omitted under clearing arrangements. Only one Return Addendum B Record is permitted for a Return Record (Type 31) and it shall must follow its associated Return Record (Type 31) or Return Addendum A Record (Type 32) when present. It is one of four addendum type records available for use with the Return Record item group.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "33".
2	Payor Bank Name	C	3	18	A	
3	Auxiliary On-Us	C	21	15	NBSM	
4	Item Sequence Number	C	36	15	NB	
5	Business Date	C	51	8	N	
6	Account Name	C	59	22	ANS	

Type 34 Return Addendum C Record

The Return Addendum C Record is conditional and is typically used to define the location of an image within an image archive. It should only be present only under defined clearing arrangements. The image archive locator record should always its immediately preceding Return Record (Type 31), a Return Addendum A Record (Type 32), or Return Addendum B Record (Type 33) when present. Only one Return Addendum C Record is permitted for a Return Record (Type 31). It is one of four addendum type records available for use with the Return Record item group.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "33".
2	Variable Size Record Indicator	M	3	1	N	0' this is an 80-byte record; Field 2 has a value of 34. '1' Field 5 is variable size.
3	Microfilm Archive Sequence Number	C	4	15	NB	

Field	Field Name	Usage	Position	Length	Format	Notes
4	Length of Image Archive Locator	M	19	4	N	Value must be 1 through 999.
5	Image Archive Locator	C	23	34	ANS	
6	Description	C	57	15	ANS	
7	User Field	C	72	4	ANS	
8	Reserved	M	76	5	B	

Type 35 Return Addendum D Record

The Return Addendum D Record represents a subsequent endorsement for this item. Presence of this record type is conditional and is used to document a specific processing entity within the endorsement chain. There may be multiple type 35 records for a given item and they immediately follow its Return Record (Type 31), Return Addendum A Record (Type 32), Return Addendum B Record (Type 33), or Return Addendum C Record (Type 34) when present. It is one of four addendum type records available for use with the Return Record item group.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "28".
2	Return Addendum D Record Number	M	3	2	N	Assigned sequentially beginning with 1.
3	Bank Routing	C	5	9	N	
4	Endorsement Date	C	14	8	N	
5	Item Sequence Number	C	22	15	NB	
6	Truncation Indicator	C	37	1	A	Y' Yes this institution truncated the original check 'N' No this institution did not truncate the original check

Field	Field Name	Usage	Position	Length	Format	Notes
7	Conversion Indicator	C	38	1	AN	'0' Did not convert physical document '1' Original paper converted to IRD '2' Original paper converted to image '3' IRD converted to another IRD '4' IRD converted to image of IRD '5' Image converted to an IRD '6' Image converted to another image '7' Did not convert image '8' Undetermined
8	Correction Indicator	C	39	1	N	0' No Repair '1' Repaired '2' Repaired without Intervention '3' Repaired with Operator Intervention
9	Return Reason	C	40	1	AN	A' NSF - Not Sufficient Funds 'B' UCF - Uncollected Funds Hold 'C' Stop Payment 'D' Closed Account
10	User Field	C	41	15	ANS	
11	Reserved	M	56	15		

Type 61 Format (001) "Metavante"

The Credit Reconciliation record type 61 format 001 is commonly used and can often be identified based on the presence of 13 fields.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value "61".
2	MICR AuxOnUs	C	3	15	NBSM	
3	External Processing Code (EPC)	C	18	1	N	
4	Payor Bank Routing	M	19	9	N	
5	MICR OnUs	M	28	20	NBSM	

Field	Field Name	Usage	Position	Length	Format	Notes
6	Amount	M	48	10	N	
7	Item Sequence Number	M	58	15	NB	
8	Documentation Type Indicator	C	73	1	AN	
9	Type of Account	C	74	1	A	
10	Source of Work	C	75	1	AN	
11	Work Type	C	76	1	ANS	
12	Debit Credit Indicator	C	77	1		
13	Reserved	C	78	3	ANS	Blanks

Type 61 Format (002) “DSTU”

The Credit Reconciliation record type 61 format 002 is commonly used and can often be identified based on the presence of 12 fields.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value “61”.
2	Record Usage Indicator	M	3	1	AN	
3	MICR AuxOnUs	C	4	15	NBSM	
4	External Processing Code (EPC)	C	19	1	N	
5	Payor Bank Routing	M	20	9	N	
6	MICR OnUs	M	29	20	NBSM	
7	Amount	M	49	10	N	
8	Item Sequence Number	M	59	15	NB	
9	Documentation Type Indicator	C	74	1	AN	

Field	Field Name	Usage	Position	Length	Format	Notes
10	Type of Account	C	75	1	A	
11	Source of Work	C	76	2	AN	
12	Reserved	C	78	3	ANS	Blanks

Type 61 Format (003) “x9.100-180”

The Credit Reconciliation record type 61 format 003 is not commonly used since it has a record length of 84 instead of the much more standard length of 80 that is shared by all x9 record formats.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value “61”.
2	Record Usage Indicator	M	3	1	AN	
3	MICR AuxOnUs	C	4	15	NBSM	
4	External Processing Code (EPC)	C	19	1	N	
5	Payor Bank Routing	M	20	9	N	
6	MICR OnUs	M	29	20	NBSM	
7	Amount	M	49	14	N	
8	Item Sequence Number	M	63	15	NB	
9	Documentation Type Indicator	C	78	1	AN	
10	Type of Account	C	79	1	A	
11	Source of Work	C	80	2	AN	
12	Reserved	C	82	3	ANS	Blanks

Type 62 Format (000) “x9.100-187-2013”

The Credit Reconciliation record type 62 format 000 was introduced as part of the x9.100-187-2013 standard and is included in x9.100-187-2016 and beyond. Note the length of this record is 100 and not 80, which makes it very different from the various type 61 credit layouts.

Field	Field Name	Usage	Position	Length	Format	Notes
1	Record Type	M	1	2	N	Value “62”.
2	MICR AuxOnUs	C	3	15	NBSM	
3	External Processing Code (EPC)	C	18	1	NS	
4	Payor Bank Routing	M	19	9	N	
5	MICR OnUs	M	28	20	NBSMOS	
6	Amount	M	48	14	N	
7	Item Sequence Number	M	62	15	NB	
8	Documentation Type Indicator	C	77	1	AN	
9	Type of Account	C	78	1	AN	
10	Source of Work	C	79	2	N	
11	User Field	C	81	16	ANS	
12	Reserved	M	97	4	ANS	Blanks