X9Ware E13B-OCR User Guide

X9Ware

Your x9.37+ACH+CPA005 support tools

Revision Date: 01/03/2024
Release R5.03
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

Visual Test Aids....3
Installation Tasks....5
JAR Requirements and ClassPath....6
SLF4J Logging....7
  Logging Frameworks....7
  X9SdkLogger....8
API Core Classes....9
X9RecoDemo – Sample Program....10
X9RecoMonitor....16
X9RecoWorker....17
X9RecoItem....19
X9Assist E13B-OCR Results Viewer....22
Submitting Problem Reports....23
Appendix *A*: MICR Line....24
  MICR Line Standards....24
  MICR Line Parsing....25
  MICR Line Characters....25
  MICR Line Fields....25
  MICR Line Layout....26
  MICR Line RegEx....27

Visual Test Aids

The X9Ware-E13B-OCR Toolkit is a proprietary optical character recognition solution designed specifically for accurate and efficient recognition of the E13B font used on checks and other financial documents. As a pure Java implementation without any JNI (Java Native Interface) dependencies, the toolkit runs natively on the Java Virtual Machine, ensuring platform independence across operating systems. The toolkit delivers fast, accurate, and reliable optical character recognition natively within the Java platform.

The X9Ware-E13B-OCR engine utilizes advanced image processing techniques to deliver robust character recognition. The image is analyzed through several phases - first the document is examined for skew and properly re-oriented, then noise removal algorithms are applied to clean the image, finally the individual E13B characters are isolated and matched against known templates. To provide ultimate performance, multi-threaded concurrent processing allows throughput to approach and possibly exceed 100 documents per second based on benchmark testing.

To match each character, both primary and secondary recognition algorithms are employed to reach high confidence results. The primary algorithm looks at the overall shape and size of the character, while the secondary algorithm is applied for additional confirmation as needed. This dual matching approach enables recognition accuracy approaching 99.8%.

Both speed and accuracy can be verified in your own environment by leveraging the built-in tools provided in X9Ware's X9Assist desktop application:

- For item level testing, the X9Assist Item Viewer displays OCR visual results for individual check images, allowing quick inspection of recognition quality. The Item Viewer can be launched first loading a file and then double clicking the front image thumbnail in the lower right corner of the X9Assist primary dashboard.
- For bulk testing, X9Assist / Utilities / Ocr / tool performs recognition of an entire X9.37 file and automatically compares the output to the original data, filtering any differences for review.
- For standalone image testing, X9Assist / Utilities / Tiff Tester / provides recognition of external TIFF files without the need for these images to be embedded within an existing X9.37 file.

We encourage you to take a look at these capabilities, using your images within your environment. We have taken the approach of implementing this demonstration tool within X9Assist, since it can be installed within your environment, on your own devices. This allows you to independently analyze as many images as needed to perform a “deep dive” into our recognition results. This means that while you are testing, images remain local and never leave your environment. We did not want to do OCR demonstrations from our website, or perhaps from a cloud server. By doing this from an X9Assist (X9Lite) freeware installation, you can perform OCR testing at your convenience while fully protecting all customer information.

In addition to being able to run OCR, the Item Viewer and Tiff Tester include the "?" button on the command line, which provides detailed information that is generated by our recognizer as it analyzes the image. This is essentially a deep dive into the process itself. You will be able to see the MICR code line that was extracted, how it was deskewed and isolated, the individual characters that were identified, and even our internal logging as the analyzer does its job. Please realize that the "?" inquiry does run much slower, since it take more resources to generate the needed tracing and logging.

By installing in your environment, you can throw difficult images at our OCR recognizer to get your own results using your "worst case" images. We encourage you to do so, comparing our results against expectations and perhaps even your current solution.

We are very interested in your comments. Please contact us and we can provide an evaluation package which will allow you to install this product within your system. For example, you can then run against hundreds or thousands of images, and compare our results against your current solution.

We always recommend evaluating the latest X9Ware-E13B-OCR Toolkit integrated in the newest X9Assist version before adoption in your own Java application. The all-Java architecture ensures seamless compatibility with your JVM environment without external dependencies. By validating performance and accuracy upfront using the X9Assist tools, you can easily verify the toolkit's capabilities on your specific document set, prior to more detailed integration testing.

Installation Tasks

X9Ware-E13B-OCR installation consists of the following basic tasks:

1. Read this guide for a full understanding of the X9Ware-E13B-OCR Toolkit.

2. Review the provided Java examples to get a better idea of how our MICR recognition can be adapted and incorporated into your environment.

3. Review the distribution materials provided within the X9Ware-E13B-OCR Toolkit 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. All required resources (including our standard MICR font) is embedded and included. There is no need for external resources.

7. Ensure you have an adequate JVM heap size set for your application based on your environmental requirements and anticipated file sizes. This is especially true if you are using the X9Ware-E13B-OCR Toolkit in multi-threading mode.

8. If you have startup problems, please provide the log and the issue description to X9Ware for our research and resolution.

9. Use our X9Ware provided Java sample 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 our SDK and the X9Ware-E13B-OCR Toolkit. 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.

Required open source jars are included in the packaged X9Ware-E13B-OCR Toolkit, as follows:

Product and Release Level	Requirement	Purpose and Comments
SLF4J: we are using 1.7.26 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 *deployment* 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 website: 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-E13B-OCR jar as packaged and distributed does not include a Java “.classpath” file. All class path requirements must be fulfilled by your “-cp” parameters.

SLF4J Logging

Logging functionality is absolutely critical to every application. X9Ware realizes that every 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-log4j12-x-x-x.jar	Binding for log4j version 1.2, 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 1.7.26)
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 1.7.26)
log4j-api-xx.jar (X9Ware currently using 2.11.2)
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();

}
```

API Core Classes

The X9Ware-E13B-OCR API consists of the following core classes:

Class	Description
X9MicrRecognition	X9MicrRecognition scans an E13B encoded micr line and extracts characters from the micr band area using image based recognition techniques. We believe that our strategy and resulting performance is unique to this solution and as such is considered highly proprietary and confidential.
X9RecoList	X9RecoList contains a list of individual blob areas on the scan line. These blob areas will initially start out to be a rendition of all blobs that are on the scan line and in some random sequence. Through a series of processing steps, we will resequence them in ascending (sorted) sequence, eliminate noise, recognize MICR characters, merge blobs that are determined to represent a single character, split blobs that are identified to represent multiple characters, discard blobs that are outside of the determined scan line area, and ultimately formulate a string of MICR line characters from this blob list. This list represents the current status of that work effort and is especially useful since it is sortable.
X9RecoMonitor	X9RecoMonitor directs image recognition activities against a series of item lists which are presented to us sequentially and processed by background threads. The exact number of started threads is dependent on the number of available processors and system property settings. Performance is maximized by splitting each input list across multiple internally created lists which are then processed independently by concurrent threads. Statistics are accumulated within each worker task and aggregated and reported on completion.
X9RecoStats	X9RecoStats defines MICR recognition statistics and provides logging services.

X9RecoDemo – Sample Program

X9RecoDemo is a sample program which runs items through the X9Ware-E13B-OCR recognition processor. This program provides an example of the built-in threading technology that is part of the recognition package. The toolkit includes standard logic that will create background threads automatically on your behalf, which will reduce elapsed time by running multiple items concurrently. If your application does not require this multi-threading capabilities, then you can simply invoke the X9MicrRecognition class directly, which will allow you to run item recognition sequentially, one item at a time.

Several notes about this example:

- As a part of your implementation, you must decide how many background threads that are to be run concurrently. This decision is based on application requirements and system available processor resources. A typical setting would be four, but it would be better to assign this via a soft parameter.
- Individual item instances are created using the X9RecoItem class. You will most probably want to create your own X9RecoItem subclass, which would allow you to store more information within the items being processed.
- The example program is designed to process all items in a high level folder, where those items can also be structured in a folder-within-folder layout. The example can also filter items based on a generate wild card pattern, which can be used to optionally filter file names and extensions. This is a powerful facility; some of these tools may be useful for your target application being developed.
- Our distribution includes a test folder that can be used to run this example.
- Recognition processing is repetitive against a list of items until all items are processed. The example program calls these work lists. This concept is necessary because attempting to process all items in a single operation could easily exceed available heap size. Because of that, the items are divided into groups (smaller work lists), with each group processed separately. Processing continues until all items have been exhausted.
- The recognition process returns the micr line, which will be blank when it could not be located. The MICR line may also contain one or more asterisks, where those represent unreadable characters. Dashes are recognized and returned. Blanks will be returned as well, with the intent that blanks are significant and returned in the fashion as they exist on the image itself.
- Each X9RecoItem contains its associated X9RecoList, which provides more information regarding the physical MICR line content and the work done by the OCR recognizer. Each localized pixel group within the isolated MICR line area is identified within this list, with attributes that describe exactly what has been found and recognized.

Here is the source code for sample program X9RecoDemoMain:

```
package com.x9ware.recognition;
```

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;

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

import com.x9ware.actions.X9Exception;
import com.x9ware.apacheIO.IOCase;
import com.x9ware.apacheIO.WildcardFileFilter;
import com.x9ware.logging.X9JdkLogger;
import com.x9ware.tools.X9BuildAttr;
import com.x9ware.tools.X9CsvWriter;
import com.x9ware.tools.X9FileIO;
import com.x9ware.tools.X9FileUtils;
import com.x9ware.tools.X9MicrLineParser;
import com.x9ware.tools.X9StringQuoted;
import com.x9ware.tools.X9ThreadPool;

/**
 * X9RecoDemoMain is a sample program which demonstrates the capabilities and usage of the X9Ware
 * E13B OCR recognition toolkit which obtains MICR lines from tiff images using OCR recognition
 * techniques. This example program reads all TIFF images from an input folder structure, which can
 * contain either images or subfolders. The folders are walked recursively to obtain all images
 * which are then analyzed currently across multiple background threads. The input folders can
 * contain both front and back side images as long as there is a naming scheme that allows the front
 * images to be identified using the image file name. This example does not read an actual x9.37
 * file since the X9Ware SDK is not included in our E13B-OCR toolkit, and hence we do not have
 * access to the tools needed to read and parse those files. The X9Ware SDK can be licensed
 * separately to obtain that functionality if needed.
 *
 * @author X9Ware LLC. Copyright(c) 2012-2018 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 final class X9RecoDemoMain {

          /*
           * Private.
           */
          private final File inputImageFolder;
          private final File outputCsvFile;
          private final List<File> inputImageList = new ArrayList<>(INITIAL_INPUT_FILE_LIST_SIZE);
          private int itemCount;
          private int listCount;

          /*
           * Constants.
           */
          private static final String X9RECO_DEMO_MAIN = "X9RecoDemoMain";
          private static final int NUMBER_OF_THREADS = 4;
          private static final int NUMBER_OF_ITEMS_PER_THREAD = 50;
          private static final int INITIAL_INPUT_FILE_LIST_SIZE = 1000;
          private static final int MAXIMUM_LIST_SIZE = NUMBER_OF_THREADS * NUMBER_OF_ITEMS_PER_THREAD;

          /*
           * MICR line symbols.
           */
          private static final char TRANSIT_SYMBOL = 'A';
          private static final char AMOUNT_SYMBOL = 'B';
          private static final char ONUS_SYMBOL = 'C';
          private static final char DASH_SYMBOL = 'D';

          /**
           * Logger instance.
           */
          private static final Logger LOGGER = LoggerFactory.getLogger(X9RecoDemoMain.class);

          /*
           * X9RecoDemoMain Constructor.
           *
           * @param input_ImageFolder input image folder to be processed
           *
           * @param output_CsvFileName output csv results file to be written
           *
           * @param inputWildCard input wild card used to select front side images
```

```
 */
public X9RecoDemoMain(final String inputImageFolderName, final String output_CsvFileName,
			final String inputWildCardString) {
	/*
	 * The E13B-OCR license XML document must be added and set here.
	 */
	// final String licenseXmlDocument = null;
	X9BuildAttr.setE13bOcrProductLicense(licenseXmlDocument);

	/*
	 * Assign files.
	 */
	inputImageFolder = new File(inputImageFolderName);
	outputCsvFile = new File(output_CsvFileName);

	/*
	 * Echo command line parameters.
	 */
	LOGGER.info("inputImageFolder({})", inputImageFolder);
	LOGGER.info("outputCsvFile({})", outputCsvFile);
	LOGGER.info("inputWildCardString({})", inputWildCardString);

	/*
	 * Ensure the input image folder exists.
	 */
	if (!X9FileUtils.existsWithPathTracing(inputImageFolder)) {
		throw X9Exception.abort("inputImageFolder not found({})", inputImageFolder);
	}

	/*
	 * Ensure the input image folder is a directory.
	 */
	if (!inputImageFolder.isDirectory()) {
		throw X9Exception.abort("inputImageFolder({}) is file not folder", inputImageFolder);
	}

	/*
	 * Build a list of all images that are found in the image folder, which can contain images
	 * or subfolders. We recursively walk the subfolders and load all images encountered.
	 */
	final WildcardFileFilter fileFilter = new WildcardFileFilter(
				X9StringQuoted.getValue(inputWildCardString), IOCase.INSENSITIVE);
	X9FileUtils.getFilteredListRecursively(inputImageFolder, inputImageList, fileFilter);

	/*
	 * Log the number of images that will be processed.
	 */
	LOGGER.info("imageFolder({})", inputImageFolder);
	LOGGER.info("number of input images({})", inputImageList.size());
}

/**
 * High level processing consists of a repetitive loop that submits throttled item lists through
 * our X9RecognitionMonitor. The lists are throttled in size to ensure the heap is not overrun.
 * The monitor will spread the items across sublists which are then routed across multiple
 * background threads for OCR recognition. The use of concurrent threads will minimize elapsed
 * time and maximum use of CPU resources. X9RecognitionMonitor can drive the CPU very high, but
 * also includes periodic pauses to ensure that CPU utilization does not reach an absolute 100%.
 * If threaded processing is not needed by the application, then X9MicrRecognition can be
 * invoked directly to process item by item.
 */
private void process() {
	/*
	 * Continuous loop until all input images are processed.
	 */
	try (final X9CsvWriter csvWriter = new X9CsvWriter(outputCsvFile)) {
		List<X9RecoItem> itemList = getNextListOfItems();
		while (itemList != null) {
			runRecognition(csvWriter, itemList);
			itemList = getNextListOfItems();
		}
	} catch (final Exception ex) {
		throw X9Exception.abort(ex);
	}
	LOGGER.info("finished; itemCount({}) listCount({})", itemCount, listCount);
}

/**
```

```
 * Run recognition against the current list of items. The items will be spread across the
 * defined number of background threads, allowing concurrent processing across the threads.
 *
 * @param itemList
 *            current item list
 */
public void runRecognition(final X9CsvWriter csvWriter, final List<X9RecoItem> itemList) {
	/*
	 * Run recognition and wait for those background threads to complete.
	 */
	final X9RecoMonitor recoMonitor = new X9RecoMonitor(NUMBER_OF_THREADS);
	recoMonitor.startWorkerThreads(itemList);
	recoMonitor.waitForCompletion();

	/*
	 * Walk the items and get individual items results.
	 */
	for (final X9RecoItem recoItem : itemList) {
		/*
		 * Parse the micr line into fields.
		 */
		final int recordNumber = recoItem.getRecordNumber();
		final String micrLine = recoItem.getMicrLine();
		final X9MicrLineParser micrParser = new X9MicrLineParser(recordNumber, micrLine,
				TRANSIT_SYMBOL, AMOUNT_SYMBOL, ONUS_SYMBOL, DASH_SYMBOL);

		/*
		 * Add this item and associated results to the output csv file.
		 */
		csvWriter.startNewLine();
		csvWriter.addField(Integer.toString(recordNumber));
		csvWriter.addField(recoItem.getImageFileName());
		csvWriter.addField(micrLine);
		csvWriter.addField(micrParser.getMicrAuxOnUs());
		csvWriter.addField(micrParser.getMicrEpc());
		csvWriter.addField(micrParser.getMicrRouting());
		csvWriter.addField(micrParser.getMicrOnUs());
		csvWriter.addField(micrParser.getMicrAmount());

		/*
		 * Write the resulting csv line to the output csv file.
		 */
		try {
			csvWriter.write();
		} catch (final Exception ex) {
			throw X9Exception.abort("csvWriter");
		}
	}
}

/**
 * Get the next list of items to be processed.
 *
 * @return next item list or null when no more exist
 */
public List<X9RecoItem> getNextListOfItems() {
	/*
	 * Allocate the next list of items to be run through OCR recognition. Our X9RecoItem class
	 * can be used as is or extended to allow it to contain user application related attributes.
	 * X9RecoItem serves as the basis of each image being process and contains both input and
	 * results. This eliminates the need to associate output back to the original input items.
	 */
	final List<X9RecoItem> workList = new ArrayList<>(MAXIMUM_LIST_SIZE);

	/*
	 * Populate the next list of items to be recognized by taking them from our input list.
	 */
	for (int i = 0; inputImageList.size() > 0 && i < MAXIMUM_LIST_SIZE; i++) {
		/*
		 * Items are removed from the input list and added to the work list.
		 */
		final File imageFile = inputImageList.remove(0);
		if (X9FileUtils.existsWithPathTracing(imageFile)) {
			/*
			 * Load the byte array for the next input image to be processed.
			 */
			final byte[] imageByteArray = X9FileIO.readFile(imageFile);
			if (imageByteArray != null && imageByteArray.length > 0) {
```

```
                                        /*
                                         * Create a new X9RecoItem instance and add to the running work list.
                                         */
                                        final X9RecoItem recoItem = new X9RecoItem(++itemCount, imageByteArray);
                                        recoItem.setImageFileName(imageFile.toString());
                                        workList.add(recoItem);
                                        if (LOGGER.isDebugEnabled()) {
                                                LOGGER.debug("adding testImage({}) imageByteArraySize({})", i,
                                                                imageByteArray.length);
                                        }
                                } else {
                                        LOGGER.error("unable to load image({})", imageFile);
                                }
                        } else {
                                LOGGER.error("image file unexpectedly does not exist({})", imageFile);
                        }
                }

                /*
                 * Log the number of items that are being submitted for the current work list.
                 */
                final int workListSize = workList.size();
                if (workListSize > 0) {
                        LOGGER.info("working on listNumber({}) size({})", ++listCount, workListSize);
                }

                /*
                 * Return the populated list of items to be run through recognition. If the populated item
                 * list is empty, then we return null to signal that we have reached the end.
                 */
                return workListSize > 0 ? workList : null;
        }

        /**
         * Main().
         *
         * @param args
         *            command line arguments
         */
        public static void main(final String[] args) {
                /*
                 * Initialize and run recognition.
                 */
                int status = 0;
                X9JdkLogger.initialize();
                LOGGER.info(X9RECO_DEMO_MAIN + " started");
                try {
                        /*
                         * Ensure that we were given three command line parameters.
                         */
                        final int argCount = args.length;
                        if (argCount == 3) {
                                /*
                                 * Run the example.
                                 */
                                final X9RecoDemoMain example = new X9RecoDemoMain(args[0], args[1], args[2]);
                                example.process();
                        } else {
                                /*
                                 * Incorrect parameters.
                                 */
                                LOGGER.warn("incorrect command line parameters; argCount({}) but expecting(3)",
                                                argCount);
                                LOGGER.warn("usage: x9recoDemo [inputImageFolderName] [outputCsvFile] "
                                                + "[inputWildCardString]");
                        }
                } catch (final Throwable t) { // catch both errors and exceptions
                        status = 1;
                        LOGGER.error("main exception", t);
                } finally {
                        X9ThreadPool.poolShutdown();
                        X9JdkLogger.closeLog();
                        System.exit(status);
                }
        }

}
```

```
		/**
		 * Main().
		 *
		 * @param args
		 *            command line arguments
		 */
		public static void main(final String[] args) {
			/*
			 * Initialize and run recognition.
			 */
			int status = 0;
			X9JdkLogger.initialize();
			LOGGER.info(X9RECODEMO + " started");
			try {
				final X9RecoDemo example = new X9RecoDemo(args[0], args[1]);
				example.process();
			} catch (final Throwable t) { // catch both errors and exceptions
				status = 1;
				LOGGER.error("main exception", t);
			} finally {
				X9ThreadPool.poolShutdown();
				X9JdkLogger.closeLog();
				System.exit(status);
			}
		}

	}
```

X9RecoMonitor

X9RecoMonitor is our standard thread monitor implementation which is based on X9TaskMonitor. We provide this source code as further insight into this processing:

```
package com.x9ware.recognition;

import java.util.List;

import com.x9ware.tools.X9TaskMonitor;

/**
 * X9RecoMonitor directs image recognition activities against a series of item lists which are
 * presented to us sequentially and processed by background threads. The exact number of started
 * threads is dependent on the number of available processors and system property settings.
 * Performance is maximized by splitting each input list across multiple internally created lists
 * which are then processed independently by concurrent threads. Statistics are accumulated within
 * each worker task and aggregated and reported on completion.
 *
 * @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 specifically restricted from other specific uses including
 *         modification, sharing, reuse, redistribution, or reverse engineering.
 */
public final class X9RecoMonitor extends X9TaskMonitor<X9RecoWorker, X9RecoItem> {

        /*
         * Public.
         */
        public int recoInspections;
        public int recoAssignments;

        /**
         * X9IqaMonitor Constructor.
         *
         * @param number_OfThreads
         *            number of threads to be utilized
         */
        public X9RecoMonitor(final int number_OfThreads) {
                super(number_OfThreads);
        }

        @Override
        public X9RecoWorker allocateNewWorkerInstance(final List<X9RecoItem> workerList) {
                return new X9RecoWorker(workerList);
        }

        @Override
        public void accumulateStatisticsFromWorkerTask(final X9RecoWorker x9recoWorker) {
                recoInspections += x9recoWorker.recoInspections;
                recoAssignments += x9recoWorker.recoAssignments;
        }

}
```

X9RecoWorker

X9RecoWorker is our standard thread worker implementation which is based on X9TaskWorker. We provide this source code as further insight into this processing:

```
package com.x9ware.recognition;

import java.awt.image.BufferedImage;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.x9ware.tools.X9TaskWorker;

/**
 * X9RecoWorker runs as concurrent thread started by the thread pool manager. We run E13B OCR
 * recognition processing against a list of items and are invoked by X9RecoMonitor.
 *
 * @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 specifically restricted from other specific uses including
 *         modification, sharing, reuse, redistribution, or reverse engineering.
 */
public final class X9RecoWorker extends X9TaskWorker<X9RecoItem> {

        /*
         * Private.
         */
        private final X9RecoStats recoStats = new X9RecoStats();

        /*
         * Public.
         */
        public int recoInspections;
        public int recoAssignments;

        /*
         * Constants.
         */
        private static final int RECO_PAUSE_INTERVAL = 10;
        private static final int RECO_PAUSE_DURATION = 10;

        /**
         * Logger instance.
         */
        private static final Logger LOGGER = LoggerFactory.getLogger(X9RecoWorker.class);

        /**
         * X9RecoWorker Constructor.
         *
         * @param sdk_Base
         *            sdkBase instance
         * @param recoList
         *            items to be analyzed
         * @param clmMap
         *            code line differences accumulation map
```

```
	 */
	public X9RecoWorker(final List<X9RecoItem> recoList) {
		super(recoList, RECO_PAUSE_INTERVAL, RECO_PAUSE_DURATION);
	}

	@Override
	public boolean processOneEntry(final X9RecoItem workerItem) {
		/*
		 * Run micr recognition for the current item and set results.
		 */
		final int recordNumber = workerItem.getRecordNumber();
		final X9MicrRecognition x9micrRecognition = new X9MicrRecognition(recoStats);
		final byte[] imageByteArray = workerItem.getImageByteArray();
		final BufferedImage bufferedImage = workerItem.getBufferedImage();
		final String micrLine = imageByteArray != null
				? x9micrRecognition.micrRecognition(imageByteArray, recordNumber)
				: x9micrRecognition.micrRecognition(bufferedImage, workerItem.getImageDpi(),
						recordNumber);
		workerItem.setMicrLine(micrLine);
		workerItem.setMicrAreaRecoList(x9micrRecognition.getRecoList());

		/*
		 * Log if debugging.
		 */
		if (LOGGER.isDebugEnabled()) {
			LOGGER.debug("finished recognition recordNumber({}) micrline({})", recordNumber,
					micrLine);
		}

		/*
		 * Update statistics and current status.
		 */
		recoInspections++;
		if (StringUtils.isNotBlank(micrLine)) {
			recoAssignments++;
		}
		currentOperation = " recordNumber(" + recordNumber + ")";

		/*
		 * Always indicate that significant work was performed.
		 */
		return true;
	}

}
```

X9RecoItem

X9RecoItem is our standard recognition class which can be extended as part of your implemenation. We provide this source code as further insight into this processing:

```
package com.x9ware.recognition;

import java.awt.image.BufferedImage;

/**
 * X9RecoItem represents a single item to be processed through E13B OCR multi-thread. This class is
 * purposefully not final, allowing it to be extended when needed to create sub-classes.
 *
 * @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 specifically restricted from other specific uses including
 *         modification, sharing, reuse, redistribution, or reverse engineering.
 */
public class X9RecoItem {

        /*
         * Private.
         */
        private final int recordNumber;
        private final int imageDpi;
        private final byte[] imageByteArray;
        private final BufferedImage bufferedImage;
        private String imageFileName;
        private String micrLine = "";
        private X9RecoList recoList;

        /**
         * X9RecoItem Constructor.
         *
         * @param record_Number
         *            input record number (used for logging only)
         * @param image_ByteArray
         *            tiff image byte array
         */
        public X9RecoItem(final int record_Number, final byte[] image_ByteArray) {
                recordNumber = record_Number;
                imageByteArray = image_ByteArray;
                imageDpi = 0;
                bufferedImage = null;
        }

        /**
         * X9RecoItem Constructor.
         *
         * @param record_Number
         *            input record number (used for logging only)
         * @param buffered_Image
         *            buffered image
         * @param image_Dpi
         *            image dpi
         */
        public X9RecoItem(final int record_Number, final BufferedImage buffered_Image,
                        final int image_Dpi) {
```

```
		recordNumber = record_Number;
		bufferedImage = buffered_Image;
		imageDpi = image_Dpi;
		imageByteArray = null;
	}

	/**
	 * Get the record number associated with this item (used for logging only).
	 *
	 * @return record number
	 */
	public int getRecordNumber() {
		return recordNumber;
	}

	/**
	 * Get the image dpi associated with this buffered image.
	 *
	 * @return image dpi
	 */
	public int getImageDpi() {
		return imageDpi;
	}

	/**
	 * Get the image byte array for this item.
	 *
	 * @return image byte array
	 */
	public byte[] getImageByteArray() {
		return imageByteArray;
	}

	/**
	 * Get the buffered image for this item.
	 *
	 * @return buffered image
	 */
	public BufferedImage getBufferedImage() {
		return bufferedImage;
	}

	/**
	 * Get the associated image file name.
	 *
	 * @return image file name
	 */
	public String getImageFileName() {
		return imageFileName;
	}

	/**
	 * Set the associated image file name.
	 *
	 * @param image_FileName
	 *            image file name
	 */
	public void setImageFileName(final String image_FileName) {
		imageFileName = image_FileName;
	}

	/**
	 * Get the micr line assigned by recognition.
```

```
	 *
	 * @return micr line
	 */
	public String getMicrLine() {
		return micrLine;
	}

	/**
	 * Set the micr line assigned by recognition.
	 *
	 * @param micr_Line
	 *            micr line
	 */
	public void setMicrLine(final String micr_Line) {
		micrLine = micr_Line;
	}

	/**
	 * Get the micr area recognition list which contains all identified pixel blobs.
	 *
	 * @return micr area blob recognition list
	 */
	public X9RecoList getMicrAreaRecoList() {
		return recoList;
	}

	/**
	 * Set the micr area recognition list which contains all identified pixel blobs.
	 *
	 * @param reco_List
	 *            micr area blob recognition list
	 */
	public void setMicrAreaRecoList(final X9RecoList reco_List) {
		recoList = reco_List;
	}

}
```

X9Assist E13B-OCR Results Viewer

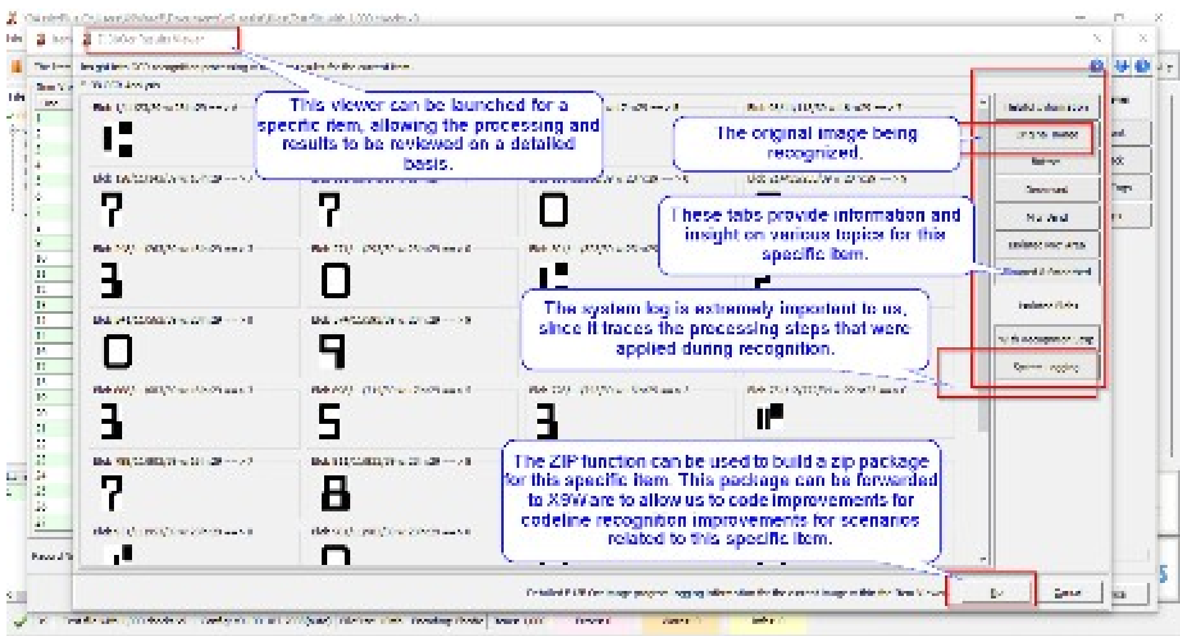
The X9Assist X9Ware-E13B-OCR Results Viewer can be launched from the Item Viewer, to provide insight into the detailed processing that was performed for a specific item. This panel not only provides this information for your review, but it also allows you to easily create a ZIP file that summarizes exactly what you are looking at.

This X9Ware-E13B-OCR results ZIP file can be provided to X9Ware to allow us to research how a specific item was analyzed and how recognition was performed. We understand that this ZIP file contains Personally Identifiable Information (PII) and will handle it in exactly that manner. We have created this ZIP facility since it allows information for a single representative item to be provided to us, instead of providing an entire X9.37 file. We do not need entire X9.37 files for this feedback cycle. However, information on single problematic images would be helpful to us.

If you forward this ZIP file to us, we will analyze it and then delete it from our systems once this research has been completed. It will not be retained in any manner.

Our X9Ware-E13B-OCR engine has been designed in such a manner that is very flexible. We can use ZIP file on our end to fully test the recognition process on our end, and to make the logic improvements that are needed to accommodate unusual situations associated with specific items. We believe that our X9Ware-E13B-OCR engine has a high level of accuracy. Your feedback can allow us to make it even better.

The X9Assist X9Ware-E13B-OCR Results Viewer appears as follows:



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 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 SDK execution. The logs are written to the system work folder unless overridden during start-up by your 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.

Appendix A: MICR Line

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-Us
- Transit
- On-Us
- Amount

All checks have at least three of the fields (amount, On-Us, and transit number). Commercial checks have an additional field on the left of the check, called the auxiliary On-Us field. Some checks also have an external processing code (EPC) digit, located between the transit and auxiliary On-Us fields. The amount and transit fields have a standardized content, while the contents of the On-Us 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

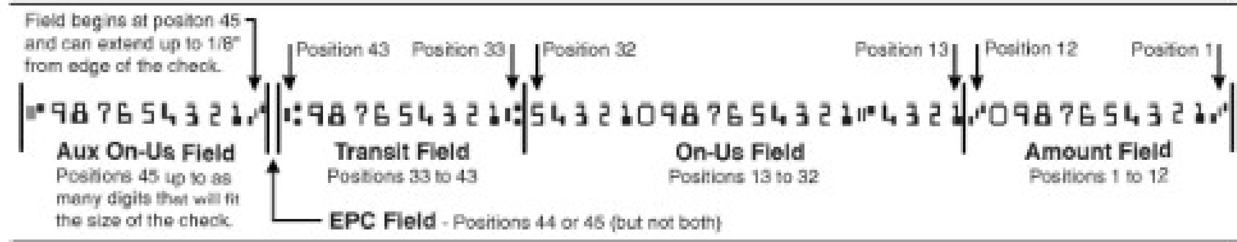
MICR Line Fields

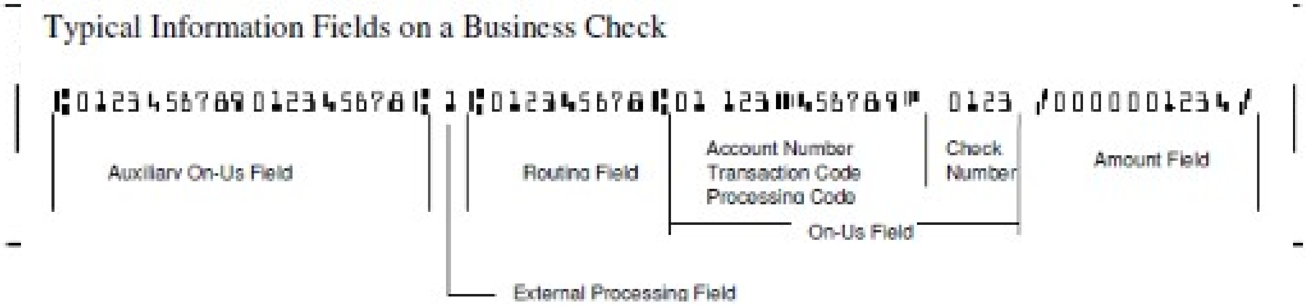
MICR line fields (from right to left on the check) are as follows:

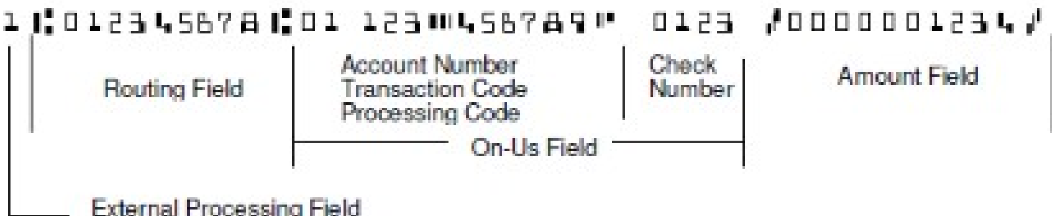
Field #	Field Name	MICR Positions	Description
1	Amount	1 – 12	Amount with leading and trailing E13B amount symbols. This field is typically not encoded in the image environment.

Field #	Field Name	MICR Positions	Description
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 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-Us 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 he 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-Us fields in position 44. When present, this field indicates that the document is eligible for special processing.
5	Auxiliary On-Us	45-65	The auxiliary On-Us 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 OnUs is not present on personal checks because of physical size of those items.

MICR Line Layout







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 AAAAAAAAAAAAAAAAAAAAAB,
- 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

<table>
<tr><th>Field</th><th>RegEx</th><th>Regex Notes</th></tr>
<tr><td></td><td></td><td>numbers and stop when it reaches anything not a number.</td></tr>
<tr><td>On-Us</td><td>(?<=A)[0-9DC]+(?=C|B|$)</td><td>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.</td></tr>
<tr><td>Transit</td><td>[0-9D]+(?=A)</td><td>Matches the part of a string that precedes A and has numbers or D.</td></tr>
<tr><td>EPC</td><td>(?<=^|B)[0-9](?=A)</td><td>Matches a single number that is preceded by B or the start of the line, and is followed by A.</td></tr>
<tr><td>Aux On-Us</td><td>[\dD]+(?=CA)</td><td>Matches the part of a string that consists of digits and D, and is followed by CA.</td></tr>
</table>

Further RegEx Reading

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

https://regexr.com/