X9Utilities User Guide

X9Ware

Your x9.37+ACH+CPA005 support tools

Revision Date: 06/15/2023
Release R5.01

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

Email support@x9ware.com

Table of Contents

X9Utilities Console

X9Vision ?	X9Validator ?	X9Assist?		X9.37 ?	ACH ?	CPA005 ?
NO	**NO**	**YES**		**YES**	**NO**	**NO**

The X9Utilities Console is an interactive tool that is available within both X9Utilities and X9Assist. The console is a powerful facility that allows command line parameters to be entered, validated, and then submitted for execution. It serves as a good demonstration of the overall capabilities of X9Utilities, since it allows the command line to be constructed visually, with validation of files and command line switches as they are entered. The console allow X9Utilities to be run on demand. It intercepts and displays the system log as x9utilities is executing. It then displays the final exit status that is posted by the x9uti.exe.

The console is an integral part of X9Utilities itself, and launched using the "-console" switch on the command line. By using this approach, it is not a separate program but instead embedded within the standard x9utilities runtime.

The console allows allows a set of parameters can be save to xml as a "run unit" that can be subsequently loaded an executed again. This is a provided as a productivity tool, since it allows a given set of parameters to be saved and executed again in the future, as needed.

As previously mentioned, X9Utilities is a command line tool. In your production environment, it will be invoked on a batch (command line) basis. This might be from a scheduled batch script, or more possibly directly by an application program. The console can serve as a workbench for your testing. It provides an illustration of x9utilities capabilities and operation.

The console is made available as part of X9Assist since that product is a desktop tool, is designed with a UI (User Interface) for human interaction, and is our full-function desktop product. X9Assist customers can use the console to actually run X9Utilities interactively in their environment. An X9Utilities license must be purchased if you subsequently decide that you would like to run the tool on batch basis.

This documentation describes how to use the X9Utilities Console. However, it does not provide detailed information on each of the functions that are available via that batch (command line) utility. Please reference the X9Utilities User Guide (which is available in its entirety via help) for complete information regarding functions, capabilities, and command line options.

X9Utilities Evaluations

The console can be used during X9Utilities evaluations, to both simplify and expedite the review process. Product evaluations are normally performed within a limited time frame and hence productivity factors are important. The console helps greatly during this process, since you can run any X9Utilites function from start to end within minutes, without the need to develop a batch script. You can also repeat the same test using a variety of input files, and can save the work unit for further repeated testing at a later time. All of these capabilities can add a lot of value during an evaluation of our X9Utilities product, so it represents a great way to get started. Once you have determined that your evaluation is favorable, X9Ware can then provide an evaluation copy of X9Utilities, which will allow you to perform the same operations on a command line basis.

X9Utilities Batch File (Windows)

The console can be invoked using a batch file. A sample of this is included as part of the Windows installation, in folder: / samples / console / .

```
@echo off

: Run the x9utilities using a start command, which allows it to run separately and the
consolse window itself to close.

cd "c:/Program Files/X9Ware LLC/X9Utilities R4.11"
start "" x9util -console

: pause

exit /b
```

Console Fields

The X9Utilities Console contains the following fields which allow you to identify the function to be performed along with their associated parameters:

- “Function” is a drop down box which allows the X9Utilities task to be selected.
- “Batch” is a check box that is used to indicate that batch processing is activate. This is a more complex runtime mode, where a command is executed against all of the files within a folder (instead of against a single file). When using batch mode, the input file must be defined as as wild-card string, where an asterisk is used for pattern matching during the file selection process. Refer to the topic “Batch/Script Operations” in the X9Utilities User Guide for more information.

- "Threads" applies only when running in batch mode, and identifies the number of threads that will be used during work unit processing.
- "Wild Card Pattern" applies only when running in batch mode, and defines the pattern that will be applied when selecting files within folders. An example might be *.x9.
- "Command Switches" is a list of switch values which are to be applied to the selected task. The entered switches must begin with a dash and must be valid for the selected function to be performed. Refer to the X9Utilities User Guide for more information.
- "Rules Specification" is used to select the x9.37 specification to be associated with the selected task. For example, the specification is critical for the "-write" since it identifies the x9 rules to be utilized, which identifies key information such as fields and field alignments.
- "Command line as executed" is a display box which summarizes the command line that will be executed for the selected task. The command line will be updated interactively as changes are made to the above fields (function, switches, rules, etc).

Input and Output Files

The X9Utilities Console includes a facility which must be used to define the input and output files that are processed. There are various files that can be specified, with certain files required based on the function being performed. These files are selected via a file chooser and the fully qualified file name is then displayed in the text box immediately to the right of each selection box. The chosen input and output files will be passed to X9Utilities via the command line. It is also informative to take a look at the command line box to see how input and output files are provided to X9Utilties via the constructed command line.

The following input and output files can be specified:

Function	Input File	Image Folder	Secondary Input File	Output File	Results File
Translate	Mandatory	Optional; will default to "_IMAGES" when not specified		Optional; is assigned based on the input file name when not specified	
Write	Mandatory		Optional; defines the HeaderXml file to be used when not provided as the first row within the	Optional; is assigned based on the input file name when not specified	

Function	Input File	Image Folder	Secondary Input File	Output File	Results File
			input csv file		
Import	Mandatory	Optional; defines the image folder base name when image file names are relative		Optional; is assigned based on the input file name when not specified	
Export	Mandatory	Optional; defines the image folder when image files are written		Optional; is assigned based on the input file name when not specified	
ExportCsv	Mandatory	Optional; defines the image folder when image files are written		Optional; is assigned based on the input file name when not specified	
Validate	Mandatory				Optional; csv of errors that were found; assigned based on the input file name when not specified
Scrub	Mandatory		Mandatory; defines the scrub xml parameters file to be used	Optional; is assigned based on the input file name when not specified	Optional; csv of fields that were scrubbed; assigned based on the input file name when not specified
Update	Mandatory		Mandatory; defines xml file which is	Optional; is assigned based on the input file	Optional output csv file name

Function	Input File	Image Folder	Secondary Input File	Output File	Results File
			used to identify the field level update parameters	name when not specified	
Split	Mandatory		Mandatory; defines xml file which is used to identify the field level split parameters	Not allowed; it is defined within the xml file.	Optional output csv file name
Make	Mandatory; defines the input use case file. Along with the input file, you must specify the "-reformatter" and "-generator" switches to define those xml files, which are mandatory as well.		Optional; defines the routing list file when it is not defined within the generator xml file.	Mandatory; defines the output file to be created.	
Merge	Mandatory; defines the input folder that contains the files to be merged.			Mandatory; defines the output file to be created.	
Compare	Mandatory		Mandatory; defines the secondary input file to be compared against		Optional output text file name
ImagePull	Mandatory	Optional; defines the image folder	Mandatory; defines csv file which is used		Optional; defines output csv file which identifies

Function	Input File	Image Folder	Secondary Input File	Output File	Results File
		when image file names are relative	to identify the item images to be pulled		the images that have been pulled.

Work Units

A work unit is a combination of all entered fields which describe the X9Utilities task to be performed. A work unit can be saved to xml and then subsequently loaded at a future time, allowing the work unit to be performed again. The load and save buttons on the right side panel are used for this purpose. Work units are stored in folder / documents / x9_assist / xml / x9utilities /. The ability to save and subsequently reuse work units allows task to be performed repetitively as needed. This might be used to perform a user periodic task and eliminates the need to create a batch script for that same purpose. Work units can not only be stored, but can also organized into subfolders within / documents / x9_assist / xml / x9utilities /.

X9Utilities Console

The console is used to display system log information for each new task that is executed. The console is initialized at the start of each new task and will be updated with log snapshots as the task is run. There are limits as to how much information can be shown within the log, which requires some truncation when the logging becomes excessive. In these situations, logging will be suspended. The console will still contain the very last lines from the log, which is important since these lines may contain output file names, totals, error messages, etc. Console content can be scrolled (using the scroll bar on the right) and can be copied to the clipboard (using CTRL A and then CTRL C). Once copied to the clipboard, the text can be pasted into other external applications as needed for specific purposes.

JDK Requirements

The console requires JDK 11 or higher for execution.

Function Execution

User input is edited interactively and errors and shown when identified. Once a valid work unit has been entered, it can be optionally saved and ultimately executed using the run button on the action line. Many tasks will run very quickly. Depending on the function performed, the number of files involved, and the relative size of those files, the run may take an extended period of time. The console has several indicators to inform you that a utility task is running:

- The background of the X9Utility console is changed from gray to cyan while the task is in a running state, and will be changed back to gray on completion.
- Exit status on the action line will be changed to “running” while the task is in a running state, and will be changed to the numeric exit status on completion.
- The status icon on the action line will be changed to a “runner” image while the task is in a running state, and will be changed back to a green/red icon on completion.

A popup message will displayed on completion that provides the exit status and the name/size of the output file. The elapsed time (in seconds) will also be displayed on the action line.

Command Line as Executed

The console includes a display box which shows the command line as executed. This is provided for several reasons:

- First is that it provides details on how the command line is formulated and thus is very helpful for exposure into the technical workings of the utility. You can compare the content of the formulated command line against the X9Utilities User Guide to fully review the command formats.
- Second is that the command line can be used as the basis for developing batch scripts. This would be done by fully testing the function to be performed, and then by creating a batch script from the command line. The new batch script may have certain parameters or file names turned into symbolic parameters, increasing both flexibility and usability.

Demonstrations

The following are examples of using the utilities console through a series of sequential operations:

##	Function	Steps needed to run the demonstration
(1)	Export x9.37 data + images into fixed column CSV format.	• Select the EXPORT function using the drop down box. • You will first need an x9.37 file to be exported. A good example would be to copy our sample file “Test ICL with 10 checks.x9” to your documents folder. You will find this x9 file in the program installation folder; for example in C:\Program Files\X9Ware LLC\X9Utilities R4.11\samples\files\. The file needs to be copied, because export will (by default) put the output CSV file and image folder next to the input file, so you need to use a folder where you have write privileges. • You will notice a series of file selection rows that are used to identify the input and output files for the current function that is to be executed. These rows select files that are automatically populated into the command line. The first of these rows is for the input file. • Once you have copied your sample x9 file to your documents folder,

##	Function	Steps needed to run the demonstration
		you then need to use the SELECT button to browse and select this as the input file. • Take a look at the switches; you can change these as desired. You will need to look at the X9Utilities User Guide for more information on the switches that are available. A good first test would be to export into fixed CSV columns with images and create a summary JSON file totals. The switches for this would be entered as: -xf -i -j • Look at the box which contains the command line as it will be executed. This shows exactly how the command line has been constructed based on your input file and entered switch values. • Hit the RUN button in the lower right to execute X9Utilities. • The X9Utilities system log will scroll as the command executes, and will eventually show as "finished" when complete. A popup box will be displayed and you can click OK. • When the export has completed, you will find the exported CSV file and the associated images located in your input folder. • Open the exported CSV file (for example, with a text viewer) and take a look at the export contents. You can use the documentation provided in the X9Utilities User Guide for a definition of the fixed format columns that have been created. • This example has assigned defaults for the image folder (secondary file) and the output file. You can alternatively assign these using SELECT. When you do that, you will also see these file names populated into the command line box.
(2)	Export x9.37 data + images into variable column CSV format.	• This is just a variation on the above example (1). The difference is that we will export into a variable number of CSV columns instead of into fixed columns. The variable column format follows the record and field definitions per the x9.37 standards. You will need to refer to the x9.37 standards themselves (eg, the 2003 DSTU) for a definition of this CSV format. • Follow the above example, but change the command line switches to indicate that variable columns should be created: -xc -i -j • Look at the box which contains the command line as it will be executed. This shows exactly how the command line has been constructed based on your input file and entered switch values. • Hit the RUN button in the lower right to execute X9Utilities. • Review output as described in the above export example.
(3)	Import x9.37 from an exported file, or a CSV of your	• This example builds on the above example (2), where the output file is in variable column CSV format. For the purposes of import, this format is mandatory, since import requires that the input file be constructed using the variable (records and fields) layout. This

##	Function	Steps needed to run the demonstration
	creation.	provides a very strict definition of the x9.37 file to be created. • Select the IMPORT function using the drop down box. • Set command line switches to write a JSON summary file and to also include additional logging: -j -l • Use the input file SELECT button to select the variable columns CSV file that was created by example (2) above. • Use the output file SELECT button to identify the output file that is to be written. For example, you can create the output file in the same folder where your input file exists. Following the example, we suggest that you create the output file as “Test ICL with 10 checks imported.x9”. • Review the selected files to ensure that they are correct. • Review the “command line” box for insight into how the import command has been constructed. • Hit the RUN button in the lower right to execute X9Utilities. • Review the created x9.37 file. You will see that it is has the exact same content as the x9.37 file that was originally exported.
(4)	Compare two x9.37 files.	• This example builds on the above examples (2) and (3), where example (2) exported a file to data+images, and example (3) then imported the data+images to create a new x9.37 file. • Given the simplistic flow of those two processes, we would expect the original x9.37 file to be equal to imported x9.37 file, which was constructed from a combined export-import process. • Select the COMPARE function using the drop down box. • Use the input file SELECT button to select the x9.37 that was input to example (2), which was the original file exported into variable column format. • Use the secondary file SELECT button to select the x9.37 file that was created by example (3), which was an import of the CSV file in variable column format. • Hit the RUN button in the lower right to execute X9Utilities. • Review the created “_output.txt” and “_output.csv” files, which will contain all identified differences. • Also take a look at the exit status that was posted by X9Utilities, which should be zero.

Reusing a Saved Work Unit

The console can be used to save a work unit to xml for subsequent reuse. This work unit can be used to the batch version of X9Utilities to rerun this command in a batch environment. This can be useful when

a function needs to be performed in a batch environment and can be repeated exactly as originally performed via the console. In some situations, this may eliminate the need to develop and test a batch script to perform the same function. The work unit is provided to X9Utilities using the **-workUnit:"-fully qualified file name"** command line parameter. No other parameters or switches are required. You can review the created xml file for more insight into the work unit xml definition and embedded content.

X9Utilities and Related Products

X9Utilities is a command line (batch) utility environment which supports a variety of functions to facilitate commonly required x9 file processing. The following specific functions are currently implemented within X9Utilities:

Product	Supported Functions		Supported Environments
X9Utilities	Write Draw Translate Export ExportCsv Import	Validate Scrub Make Update Split Image Pull	Windows, Linux, OSX.
X9Export	Export ExportCsv Translate		Windows, Linux, OSX.

X9Utilities is designed with a simple CSV interfaces for complex operations such as x9 file write.

X9Utilities provides an extremely high level of functionality for x9 users who need to write, read, export, import, validate, and scrub x9 files from their internal applications environment, but do not require the more extensive capabilities and complexities that are typically associated with using an SDK based product. Why acquire and use more complex and expensive tools than you really need? If you need to read and write x9 files, then X9Utilities is the best tool for the job at hand.

X9Utilities will do all the heavy lifting on your behalf. It uses our SDK and assumes all responsibility for field alignment and padding based on each x9 field definition, per the x9 standards. Images are converted as needed to be TIFF x9.100-181 compliant.

X9Utilities is full function, easy to use, includes our standard support, leverages our SDK, and is very competitively priced when compared to other alternatives within the x9 marketplace.

X9Utilities is designed to process both ICL and ICLR files. It supports a wide variety of x9 record types including credits, addenda records and type 68 user records.

X9Utilities is targeted for those organizations who need access to their x9 data (records and tiff images) that are embedded within x9 files. X9Utilities provides the power you need to both read and write these files using your languages such as VB, C++, Java, PERL, RUBY, Python, or similar tools. Anything that can read and write CSV files can work with X9Utilities.

X9 Image Exchange Standards

There are many standards that are associated with x9 image exchange which are maintained by several standards committees:

The standards body for Financial Services is Accredited Standards Committee (ASC) X9, Inc. which administers the American National Standards (ANS). This group defined the initial DSTU x9.37 standard in 2003 as part of the initial implementation of electronic check image exchange. Standards for this group can be referenced at:

http://www.x9.org

The American National Standards Institute (ANSI) group has subsequently defined various industry standards for image exchange. This includes updated x9.37 exchange standards (x9.100-187-2008 and x9.100-187-2013) as well as TIFF exchange standards (x9.100-181). Standards for this group can be referenced at:

http://www.ansi.org/

The DSTU x9.37 standard has been retired since has been replaced with the newer x9.100-187. We will not attempt to provide a link here, but you can easily find this PDF document with a few Google searches.

Major financial institutions and third-party processors have banded together to define what is termed as the Universal Companion Document (UCD). This document provides excellent information on how the x9.100-187-2008 standard has been implemented by a large subset of processors. To quote their website: “The purpose of this document is to formalize an industry standard for check image exchange using the ANS X9.100-187-2008 standard format and a compilation of industry best practices. This document is not intended to replace the ANS X9.100-187-2008 standard, but rather to clarify how financial institutions should use the standard to ensure all necessary and appropriate payment data is exchanged between collecting and paying institutions.:

You can reference this copy of the UCD on the Check Image Central website at:

http://www.checkimagecentral.org/

Windows EXE Batch Scripts

In the Windows environment, a batch script (also known as a batch file) is a system text file that contains instructions which are used to generate actual commands to be executed. A Windows batch file typically has the ".bat" extension and is invoked from the command prompt. When you type the batch file name at the command prompt, the Windows command interpreter (Cmd.exe) will run the specified commands sequentially as they appear within the batch file. Comments may be included in your batch file and are identified with a leading ":". Batch scripts may contain blank lines as needed for clarity.

These sample batch scripts have been developed for "-write" usage. However, the concepts presented here can be used as a basis for other similar functions. They are being provided on an informational "as is" basis. The concepts presented here are provided as examples only, with the understanding that they would need to be modified to meet specific customer needs. As with all examples, you will need to thoroughly test any resulting batch file for applicability to your specific environment. The batch files are provided here and also in the x9utilities distribution.

x9writer.bat – example #1

A simple batch script which runs X9Utilties on Windows is as follows:

The batch file is:

```
@echo off

: Run x9utilities using the "-write" function to create a new x9 file from an input csv.

: %1 is the headerXml file
: %2 is the inputCsv file
: %3 is the outputX9 file

cd "c:/Program Files (x86)/X9Ware LLC/X9Utilities Rx.xx
x9util -write -j -l -xml:"%1" "%2" "%3"

exit /b
```

Which could be invoked as follows:

```
@echo off
: Invoke x9utilities using the x9writer.bat script.
set "headerXml=c:/users/x9ware5/documents/x9_assist/files_Utilities/x9headers.xml"
set "inputCsv=c:/users/x9ware5/documents/x9_assist/files_Utilities/test1000Reader.csv"
set "outputX9=c:/users/x9ware5/documents/x9_assist/files_Utilities/test1000Reader writer2.x937"

call "c:/users/x9ware5/dropbox/batchFiles/x9writer.bat" %headerXml% %inputCsv% %outputX9%
exit
```

x9writerProcessor.bat – example #2

This more complex example is designed to scan an input folder and run x9utilities against an input csv file folder and create output x9 files. The process is designed to be invoked from an application environment but could also be trigger externally from the Windows Scheduler. The design is based on the following objectives:

- Process all csv files that are present in the input csv folder.
- Run x9utilities for each csv file that is present.
- Ignore all non-csv files that are present in the input folder.
- Move each csv file to a time stamped processed folder, which will prevent it from being selected in the next processing run. This move is only done when the output file has been created in the output folder, which indicates that the operation has been successful.

The batch file is:

```
@echo off

: Process all CSV files from a folder through x9utilities.
: Each file is moved to a date stamped processed folder so it can only be run once.
: Process folders are date stamped to allow them to be retained for future reference.

SETLOCAL ENABLEEXTENSIONS
set "csvExtension=csv"
set "x9Extension=icl"
set headerXml="c:/users/x9ware5/documents/batTest/x9headers.xml"
set "csvInputFolder=c:/users/x9ware5/documents/batTest/input"
set "outputFolder=c:/users/x9ware5/documents/batTest/output"
set "processedFolder=c:/users/x9ware5/documents/batTest/processed"

: Assign output file timestamp from the current system date and time, in YYYYMMDD_HHMMSS format.
set "sysdate=%DATE: =0%
set "systime=%TIME: =0%
set "currentYYYYMM=%sysdate:~10,4%%sysdate:~4,2%"
set "currentYYYYMMDD=%currentYYYYMM%%sysdate:~7,2%"
set "currentTime=%systime:~0,2%%systime:~3,2%%systime:~6,2%"

: Process all CSV files that currently exist within the csv input folder.
setlocal EnableDelayedExpansion
for %%a in ("%csvInputFolder%\*.%csvExtension%") do (
   : Get the csv file name which excludes the leading path name and the trailing file extension.
   set "csvName=%%~na"

   : Assign output file name and trim any resulting leading or trailing spaces.
   set outputFileName="!outputFolder!/!csvName!_!currentYYYYMMDD!_!currentTime!.!x9Extension!"
   CALL :TRIM outputFileName

   : Invoke x9utilities to process this specific (individual) file.
   cd "c:/Program Files (x86)/X9Ware LLC/X9Utilities Rx.xx"
   x9util -write -j -l -xml:"!headerXml!" "%%a" "!outputFileName!"

   : Append the csv file modification YYYYMM as a suffix to create the processed folder name.
   set "csvProcessedFolder=!processedFolder!_!currentYYYYMM!

   : Allocate the output folder when it does not currently exist.
   if not exist "!csvProcessedFolder!" md "!csvProcessedFolder!"
```

```
   : Move this csv file from the input folder to the processed folder to eliminate it from any
future processing.
   if exist "!outputFileName!" move "%%a" "!csvProcessedFolder!/!csvName!.%csvExtension%"
)

pause

exit /b

:TRIM
SetLocal EnableDelayedExpansion
Call :TRIMSUB %%%1%%
EndLocal & set %1="%tempvar%"
GOTO :EOF

:TRIMSUB
set tempvar=%*
GOTO :EOF
```

Batch Image Conversions

Images must be provided to X9Utilities in standard exchange format per the x9.100-181 standard, which can be purchased at https://webstore.ansi.org . This standard defines the TIFF tags that must be used when generating the black-white (binary) images that are required per x9.37.

All commercial scanners have the ability to generate images in this format, given that these vendors understand this requirement and thus generate images accordingly.

Part of the standard is that images must be presented at either 200 or 240 DPI. It is important to note that it is not acceptable to capture images at some other lower DPI (perhaps 96) and then resize and upscale the images to 200 DPI. Although this technically accommodates the requirement, it does not capture and forward the image at the density that downstream processors will need for their subsequent processing. Remember that the captured images are going to be used by automated applications for detailed image analysis, and also that they will be stored in image archives for a minimum of seven years. The bottom line is that it is mandatory that images be scanned at 200/240 DPI by the original capture device.

Once a 200/240 DPI image exists, it is then possible to translate that image from the original format to an x9.100-181 TIFF compatible image. If this cannot be done by the capture device, it can be subsequently converted by a large variety of tools including ImageMagick, IrfanView, LibTiff, JAI, and many others. Imaging tools can be incorporated directly into the capture process, or the images can be converted as part of a later processing flow.

It is possible to apply batch conversion of images from one format to another, where the input and output images are stored at the folder level. In other words, a folder of images are converted from some format (for example, PNG or TIF) to the x9.100-181 TIFF format. The following is an example of running IrfanView in command line (batch) mode to do this conversion at a folder level, which will read all input images and write them to an output folder:

```
:: IrfanView batch conversion - folder to folder.
:: Input can be almost any image format -- output will be x9.100-181 compliant tiff.
```

```
:: %1 is the input folder terminating with wildcard (eg, *.tif).
:: %2 is the output folder terminating with wilcard (eg, *.tif).
:: Input images must be dpi 200 or 240, since IrfanView will not change the image dpi.
:: There is a "/dpi=" command, but that refers to the dpi when scanning (not converting).

cd "c:\Program Files\IrfanView"
i_view64.exe %1 /bpp=1 /tifc=4 /convert=%2

exit /b
```

Emails on failures

In production environments it is often beneficial to include error reporting via emails. Although there is not standard support for email origination in Windows batch, there are various mechanisms and tools that can be used for this. One such freeware tool is Blat, which is typically used for this purpose. The following is an example of blat email origination based on the error level.

```
if %errorlevel% lss 0 (
   set to = "-to eMail\@MyDomain.com"
   set f = "-f eMail\@MyDomain.com"
   set server = "-server smtp.MyDomain.com"
   set subject = "-subject \"Testing blat\""
   set body = "-body \"Testing blat\""
   set debug = "-debug"
   blat - %to% %f% %subject% %body% %server% %debug%
)
```

Java JRE and Command Line Scripts

X9Utilities can be run on either the Oracle or OpenJDK at version 8 or higher. Our development environment currently runs on Zulu-8 (OpenJDK).

The following is an example of running the X9Utilities JAR in a Windows JVM environment. This sample can be used as a model for various Linux and Unix environments. The batch script is as follows:

```
@echo off
set inputFile="c:/users/x9ware5/documents/x9_assist/files_Utilities/Test file
with 2 checks reader.csv"
set outputFile="c:/users/x9ware5/documents/x9_assist/files_Utilities/Test file
with 2 checks writer.x937"
cd "c:/Users/X9Ware5/X9WareDrive/X9Utilities"
java -Xmx512m -jar x9utilities_4.xx_yyyymmdd.jar -write -j -l  %inputFile%
%outputFile%
pause
exit
```

Using a Fully Qualified “java.exe” Reference

On most environments, there may well be multiple versions of the Java Runtime Environment (JRE) installed. When you execute “java” command, it will run that version of the Java JVM that has been assigned as the system default, which typically will be the most recent version of the Java JRE that has been installed. However, be advised that this may or may not be what you expect. X9Ware requires Java SE 1.8 or higher. Most specifically, you cannot use Java EE (Enterprise Edition) due to potential class loading issues. If you have multiple versions of Java installed on your workstation, then we highly recommend that you fully qualify the “java.exe” command reference. For example,

```
“c:/Java/jre1.8.0_171/bin/java.exe” -Xmx512m -jar
x9utilities_4.xx_yyyymmdd.jar -write -j -l  %inputFile%
```

System Log

X9Utilities is built on top of the X9Ware SDK where logging is implemented using SLF4J. X9Utilities is then bound to the JDK logger as an actual implementation.

X9Utilities creates a new time stamped logging file as part of each run. The logging file can be very helpful in numerous situations:

- The command line is logged which can be helpful to confirm that you batch script is providing the command line as expected.
- The "-l" command line switch can be used to adding record level logging during development and debugging.
- Input and output file names are logged which can serve as an audit trail for each run.
- The release and build level are included as informational items.
- Program exceptions are logged which can be provided to X9Ware LLC should you encounter application issues.

By default, system log files are written to Windows folder / AppData / X9Ware LLC / log / and to Linux folder / home / x9_ware / log/. You can override this default folder location through use of the "-log" command line switch which specifies the folder location where logs should be created and written. An example of this parameter is -log:"c:/userFolder/logs/".

X9Utilities will retain log files for up to three days after which they are automatically deleted. You can adjust this retention period by providing an "options.xml" file which indicates the number of days that you would like the logs retained. You can use X9Assist or X9Validator to create your xml options file.

A sample system log is as follows:

```
X9Utilities started; logFolder[defaulted] [11:24:34.615]
systemLog(C:\Users\X9Ware5\Documents\x9_assist\log\x9assist_20161109_112434_594.log) [11:24:34.628]
command line: -t -x -l -write c:/users/x9ware5/documents/x9_assist/files_Utilities/Test file with 2 checks writer.csv
c:/users/x9ware5/documents/x9_assist/files_Utilities/Test file with 2 checks writer.x937 [11:24:34.629]
console window can be enabled using x9utilConsoleModeOn.bat and disabled using x9utilConsoleModeOff.bat [11:24:34.632]
input file(c:\users\x9ware5\documents\x9_assist\files_Utilities\Test file with 2 checks writer.csv) [11:24:34.634]
output file(c:\users\x9ware5\documents\x9_assist\files_Utilities\Test file with 2 checks writer.x937) [11:24:34.634]
properties defaulted [11:24:34.637]
launch folder defaulted from absolute path(C:\Users\X9Ware5\X9WareGitRepository\x9Utilities) [11:24:34.638]
homeFolder set from FileSystemView(C:\Users\X9Ware5\Documents\x9_assist) [11:24:34.638]
X9Utilities SDK release(R3.06) build(2016.11.07) all supporting materials intellectual property of X9Ware LLC [11:24:34.639]
systemdrive(/c) user(X9Ware5) javaHome(C:\Program Files\Java\jre1.8.0_91) javaVersion(1.8.0_91) isJetCompiled(false)
[11:24:34.640]
startup environment: Java is64Bit(true) total memory(125,952k) free memory(119,286k) [11:24:34.641]
….
….
all image readers closed [11:24:35.188]
thread pool closed; largestPoolSize(3) completedTaskCount(3) activeThreads at shutdown(0) [11:24:35.189]
shutdown complete [11:24:35.189]
X9Utilities exitStatus(0) [11:24:35.189]
log closed [11:24:35.190]
```

Console Logging

X9Utilities is distributed with console logging active (in additional to file logging). The console window is opened immediately as part of initialization. Messages are displayed within this scrolling window as the utility executes. Further considerations are:

- The console window will remain open as long as X9Utilities is running.
- The console window will be closed when X9Utilities is completed and not paused by a batch script.
- All message levels are written to the console window (it is a full copy of what is written to the system log). This includes possible DEBUG level messages which may be enabled.
- When X9Utlities is being run from a batch script, you can include a PAUSE statement that will allow the console window to remain open (at the pause point). You can then use the enter key to allow the batch script to continue and ultimately complete.

In some environments, it may be desirable to hide the console window. For example, you may want to turn off console logging in a production-based server environment. Keeping the console window enabled does not create issues. However, since the console window is not needed, it may be a distraction and can be easily turned off.

X9Utilities console mode is controlled by the following command line switches:

- “-consoleOn” is used to enable the console, which is the default when omitted.
- “-consoleOff” is used to disable the console.

System Log Correlation

Each execution of X9Utilities creates it's own unique system log using a time-stamped file name, where each log file is located in the / AppData / Roaming / X9Ware LLC / log / folder. This default folder location can be changed using the "log:" command line switch.

In more complex environments, it may be helpful to be correlate the system log from each X9Utilities run back to specific application processing or logical events. This need and implementation is customer specific. The facility is used as part of external processing of the system log, perhaps using automation / aggregation tools such as Splunk. In these situations it may be useful to be able to correlate a given system log against expected events that are being initiated. For example, a line can be inserted into the system log when a given file is processed, which can then be matched against file origination. Similarly, individual runs can be identified and as an index for errors / exceptions that are detected.

In support of this, X9Utilities has several facilities that allow user lines to be inserted directly into the system log. These are as follows:

- The "-logger:" command line switch allows user defined text to be written directly to the system log. Care should be used to include this logging string with quotes, given that there may be embedded blanks. An example might be: -logger:"run 56893".

- Additionally, the "-writer" function supports the a "logger" csv line type which can be used in a similar fashion to writer user defined text directly to the system log. Since this line is included within the incoming csv file, the content can be at a very detailed level. For example, the logging line might identify a specific deposit that could be matched via against an application origination event. An example might be: logger,"deposit 4199306".

Please let us know if there are enhancements or extensions that would provided improvements to these log correlation facilities.

Command Help

X9Utilities can provide command level help when the console window is enabled. This facility can be used to easily obtain a quick description of each X9Utilities command and eliminate the need to reference the X9Utilities User Guide.

A more detailed list of help information for X9Utilities commands can be obtained as follows:

```
=====================================================================
command usage:
x9util -console -h
x9util -batch -h
x9util -script -h
x9util -write -h
x9util -draw -h
x9util -translate -h
x9util -import -h
x9util -export -h
x9util -exportCsv -h
x9util -validate -h
x9util -make -h
x9util -merge -h
x9util -update -h
x9util -split -h
x9util -compare -h
x9util -scrub -h
x9util -imagePull -h
-h provides more detailed information for each of the above functions
=====================================================================
```

For each X9Utilities command, you can then also ask for more detailed help. For example, you can enter “x9util -write -h” which provides the following output:

```
=====================================================================
command usage:
x9util -write inputFile.csv [outputFile.x9]
[-config] [-l] [-j] [-x] [-t]
writes a new x9 output file from the provided input csv file
all image filenames must be provided in absolute format
outputFile      defaults to inputFile.x9 when not specified
-config         specifies the x9 configuration to be loaded
-l              lists all records to the log
-j              creates the json totals file in the output folder
-x              creates the xml totals file in the output folder
-t              creates the text totals file in the output folder
=====================================================================
```

Exit Codes

X9Utilities utilizes standard exit status codes, also known as exit status or error level, to convey information about the outcome of program execution. These exit codes are numeric values that are posted upon program completion.

In the context of X9Utilities, exit codes are represented as 32-bit signed integers. Conforming to general conventions, a positive exit status signifies successful execution, while a negative exit status indicates an error or failure.

X9Utilities will assign positive exit codes to indicate that processing was successful and output file(s) have been created. Alternatively, X9Utilities will assign negative exit codes to indicate that there has been some type of failure and that processing has been aborted.

When the exit code is non-zero (either greater than zero or less than zero), the system log will contain messages detailing the encountered conditions.

During program completion, many X9Utilities functions will rename their output files from "TMP" to the final filename, but this renaming action only occurs when the exit code is positive.

X9Utilities employs standard exit codes across all functions to denote absolute success (zero) or absolute failure (negative). Additionally, depending on the specific function being executed, additional positive exit codes may be used. The table below outlines the standard zero and negative exit codes used by X9Utilities:

Exit Code	Usage
0	Execution was successful.
-1	Execution was aborted due an internal error. Refer to the created system log for more information. If you are unable to resolve the issue, please provide the system log and any other supporting information to X9Ware to allow us to resolve the issue.
-2	Invalid or missing function was provided on the command line. Refer to the system log which includes an echo of the command line exactly as provided.
-3	Input file for the selected function was not found. The system log includes a path trace which provides additional information on the file search that was performed and where the not found condition was identified within the file system.

Batch / Script Operations

Batch and script operations are available to meet the needs of more advanced processing requirements. The usual X9Utilities run relies on a command line that contains the switches and file names that are needed to invoke a specific process. Alternatively, batch and script operations build a sequence of work units to be executed on top of command line processing, allowing a single X9Utilities run to process multiple files. There are two types of operations:

- The "-batch" command line switch triggers batched operations. In this scenario, a standard command line with the "-batch" switch is provided as a model to create a series of work units. The input file on this command line contains a wild card pattern (e.g., using an asterisk with the file name). Batch operations identify all files within the input folder that match this wild card pattern and create work units for each file. For example, the input command line may be expanded into hundreds of work units depending on how many files match the pattern.
- The "-script" command line switch starts scripted operations. In this case, the input file is a text (txt) file that contains the command lines to be executed. Script operations read the indicated text file and generate a run unit for each line contained within the script. This processing is similar to batched operations but provides even more control over the command lines to be executed since they can be created by your customized scripting process.

The “-threads:n” command line switch is used to optionally define the number of background threads that will be used for processing. The default is no threading, which will result in the work units being run sequentially. When threading is initiated, the work units are then assigned to background threads and will be run in a more random order. Threading will reduce the overall elapsed time, given that processing will be run concurrently. It has the side effect that a higher level of CPU time will be used as the work units are executed, to the detriment of other applications running within this same environment. All processing will be recorded within a single system log. Care should be taken to sure that all needed resources, and especially virtual memory, are made available to x9utilities.

The “-aoe:n” command line switch is used to optionally indicate that overall execution should be aborted when there are some number of work units that were aborted, where this is identified by a negative exit status. For example, the “-aoe” switch indicates that overall execution should end (as soon as possible) when an exception has occurred. In this situation, although no new work units will be started, those that are currently executing will continue to their completion. All subsequent work units will be flushed and not executed. Another example would be “-aoe:2”, which would allow one exception but would begin to flush work units after there are two exceptions.

In support of batch operations, there

Parameter	Description	Example
#fn#	File name.	C:\Users\X9Ware5\Documents\x9_assist\files\Test ICL with 100 checks.x9
#fp#	File path.	C:\Users\X9Ware5\Documents\x9_assist\files\
#fpns#	File path no separator.	C:\Users\X9Ware5\Documents\x9_assist\files
#fnnx#	File name no extension.	C:\Users\X9Ware5\Documents\x9_assist\files\Test ICL with 100 checks
#fnb#	File name base.	Test ICL with 100 checks
#fnx#	File name extension.	x9
#fnbx#	File name base with extension.	Test ICL with 100 checks.x9
#i#	Work unit number.	5
#i2#	Work unit number as 2 digits.	05
#i3#	Work unit number as 3 digits.	005
#i4#	Work unit number as 4 digits.	0005
#i5#	Work unit number as 5 digits.	00005
#i6#	Work unit number as 6 digits.	000005
#yyyyMMdd#	Date formatted as "yyyyMMdd".	20230426
"#kkmmss#"	Time formatted as "kkmmss" which is based on the 24-hour clock.	220418
#HHmmss#	Time formatted as "HHmmss" which is based on the 12-hour clock.	100418
#kkmmssSSS#	Time formatted as "kkmmssSSS" which is based on the 24-hour clock and includes milliseconds.	220418521
#HHmmssSSS#	Time formatted as "MMmmssSSS" which is based on the 12-hour clock and includes milliseconds.	100418521
#yy#	Date as "yy".	23

#yyyy#"	Date as “yyyy”.	2023
#MM#	Date as “MM”.	04
#dd#	Date as “dd”.	18
#DDD#	Date as “DDD” which is the julian date within the year.	257
#kk#	Time as hour from 24-hour clock.	22
#HH#	Time as hour from 12-hour clock.	10
#mm#	Time as minutes.	04
#ss#	Time as seconds.	18
#SSS#	Time as milliseconds.	521

An example of batch operations is as follows:

```
-batch
-threads:8
-scrub
"C:/Users/X9Ware5/Documents/x9_assist/imagePull_Testing/file*.x9"
"C:/Users/X9Ware5/Documents/x9_assist/xml/scrub/scrubParameters.xml"
"C:/Users/X9Ware5/Downloads/threadTesting/#fnb#_scrubbed.x937"
"C:/Users/X9Ware5/Downloads/threadTesting/scrubbedFields.txt"
```

An example of script operations is as follows:

```
-script "C:/Users/X9Ware5/Documents/x9_assist/test/translateAndWrite.txt"
```

Write and Translate

Reading and writing x9 files can quickly become a complex process:

- There are a large number of record types and data fields, all of which have very specific value and alignment requirements.
- There are variable length records (especially for images) that must have their data elements and lengths correctly formatted to allow the file to be parsed.
- There are multiple formats associated with the insertion of credits, since they were not covered by the early x9.37 standards. This impacts the credit format, where they are inserted into the x9 file, and their impact on trailer record counts and amounts.
- Most x9 files must be created using the EBCDIC character set (not ASCII) which further complicates the creation process.
- Finally and most importantly, many financial institutions have implemented their own x9 file variant requirements, which further complicate an already difficult situation.

The Write and Translate functions are designed with the purpose of making this process as easy as possible. By using these tools, you can concentrate on the actual x9 data content and not the underlying complexities associated with x9 files.

Write is one of the most powerful x9 file creation tools that you will find anywhere in the marketplace today, and has been designed to be easily incorporated into your application environment. Write creates x9 files from a simple CSV (which defines your items) and a statically defined HeaderXml definition (which defines the x9 file attributes). Although the CSV can be provided in multiple formats, our recommended format has a single row per item, allowing it to be easily created and viewed. The HeaderXml file (see the appendix) has 100+ parameters which allows the x9 file to be written specifically to the requirements provided by your financial institution in their companion document. All of the complexity of adapting to those requirements are satisfied through the HeaderXml definition. Write is simple, straight forward, and very easy to use.

Translate is provided within X9Utilities for completeness, but in reality will be seldom used in most environments. Translate creates a CSV file in a format that can be used as input to write (they share a common CSV format). In this way, Translate can be used to create a sample CSV from an existing x9 file to provide some insight into the process. However, in the real world, our Export function would more typically be used to convert a x9 file to an output CSV, since the entire definition of the x9 file on a record and field basis is fully retained when using Export.

The CSV file begins with an optional header line and is terminated with an “end” line, as follows:

> headerXml parameter line (optional since this can also be provided on the command line)
> zero or more items
> end

Several notes about the CSV file format are as follows:

- Although the first line of the CSV file can point to the HeaderXml file to be used by Write, this is optional since that directive can instead be provided as a command line parameter using the (“xml:” command line option). See the appendix for a full definition of available HeaderXml file content.
- A zero items CSV file is totally valid and will result in the creation of a file with record types 01, 10, 90, and 99.
- Quotes around numeric fields are optional and never required.
- Quotes around alpha fields are only required when the value contains embedded blanks (since a blank will be considered the end of the input line if it is encountered outside of a quoted field). Make sure you include quotes around field values that potentially can contain blanks. This is especially true of file names.
- Our recommended format for Write uses a “t25” line that contains both item and image information. The remaining information is specified in the HeaderXml definition, which allows the x9 file to be constructed as required by your financial institution. HeaderXml includes various formatting instructions, indicators, addenda record information when needed, and instructions on how and where an offsetting credit should be inserted. See the appendix for a full definition of available HeaderXml file content.

A sample CSV file is as follows, which would default all type 25 record indicators to the HeaderXml file:

```
t25,10002,44000001,087770706,"29602722/5526",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
t25,10004,44000002,097770592,"60333044/5587",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
t25,10006,44000003,077770392,"29343913/5178",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
end
```

Export and Import

Export and Import support multiple formats which are flexible to meet your specific needs. You can work directly with your x9 files within their native record and field definitions, giving you direct access to their encoded data. Export also has more simplified formats that are parsed into fixed columns, which makes it very easy to work with that data. These various formats are easy to use and are recommended over our “-read” format, which is typically used only as a companion tool for our “-write” function and is very proprietary to X9Ware.

Export supports multiple output formats (CSV, XML, and text). The most powerful format is CSV export into fixed columns, which exports each logical item as a single row with all associated data elements parsed and written within the export utility. This allows your application to readily process the data since all fields will be constant positions which simplifies your reformatting substantially. You can optionally export the images when exporting into fixed columns, making it useful in a wide variety of application situations.

As an alternative, Export and Import also support a CSV format that exactly matches the records and fields that are defined within a x9 file (this is referred to as native format). When using this specific format, Export and Import are complimentary tools. Export will read a x9 file and create a CSV file with records formatted exactly as they appear within the x9 file. Import takes a similarly formatted CSV file and creates an output x9 file.

Both of these tools allow you to specify the x9 configuration via the command line. This is an important consideration, since the fields within a x9 record may potentially vary based on the x9 specification that is being used.

For example, the following sequence will take take an existing x9.37 file, export it to CSV, and then import the CSV back to x9.37. The original file and imported file will be equal.

The export definition:

```
-l -t -i
-export
"c:/users/x9ware5/documents/x9_assist/files_Utilities/Test ICLR with 10 checks.x9"
"c:/users/x9ware5/documents/x9_assist/files_Utilities/Test ICLR with 10 checks.csv"
```

The import definition:

```
-l -t
-import
"c:/users/x9ware5/documents/x9_assist/files_Utilities/Test ICLR with 10 checks.csv"
"c:/users/x9ware5/documents/x9_assist/files_Utilities/Test ICLR with 10 checks.x937"
```

A sample CSV file in fixed column format is below where the item rows begin with “25” and the column data is per our export documentation that you will find within the export topic.

```
01,03,"P",123456780,123456780,20141017,1201,"N","Test File","Test File","A",,,
10,01,123456780,123456780,20141016,20141017,1201,"I","G",1,"X9ASSIST",,"C",,,
20,01,123456780,123456780,20141016,20141017,57000000,1,,123456780,,
25,10000,44000000,057770930,"20915353/7837",,,"G",8,1,"Y",03,0,"B",,,,,,,,,20915353,7837,
,,,20141014,057770930,"TEST
KEY",,,26,123456780,20141014,44000000,,,,"Y",0,,,,,28,231382458,20141015,1,,,,"N",0,0,,,,
28,221374984,20141015,2,,,,"N",0,0,,,
25,10002,44000001,087770706,"29602722/5526",,,"G",8,1,"Y",03,0,"B",,,,,,,,,29602722,5526,
,,,20141014,087770706,,,,26,123456780,20141014,44000001,,,,"Y",0,,,,,28,231379636,2014101
5,3,,,,"N",0,0,,,,28,101103152,20141015,4,,,,"N",0,0,,,
70,0002,000000020002,000000020002,00004,,
90,000001,00000002,00000000020002,000000004,"File Generator",20141017,
99,000001,00000022,00000002,0000000000020002,,,
```

A sample CSV file in native format is as follows:

```
01,03,"T",123456780,123456780,20140810,1201,"N","VIEW","VIEW","A",,,
10,01,123456780,123456780,20140808,20140810,1201,"I","G",1,"X9Assist",,"C",,
20,01,123456780,123456780,20140808,20140810,57000000,1,,123456780,,
25,,,08777070,6,"29602722/5526",0000010002,44000001,"G",8,1,"Y",01,0,"B"
26,1,123456780,20140807,44000001,,,,"Y",0,,,
50,1,087770706,20140807,00,00,0006302,0,00,0,,,,,0,,
52,123456780,20140808,,44000001,,,,0,,,,,0000,,0,,0006302,"c:/users/x9ware5/documents/
x9ware/files_Utilities/Test file with 2 checks
exporter_IMAGES/Bundle_000003/Image_000007_amount_10002_front.tif"
50,1,087770706,20140807,00,00,0001865,1,00,0,,,,,0,,
52,123456780,20140808,,44000001,,,,0,,,,,0000,,0,,0001865,"c:/users/x9ware5/documents/
x9ware/files_Utilities/Test file with 2 checks
exporter_IMAGES/Bundle_000003/Image_000009_amount_10002_back.tif"
25,,,09777059,2,"60333044/5587",0000010004,44000002,"G",8,1,"Y",01,0,"B"
26,1,123456780,20140807,44000002,,,,"Y",0,,,
50,1,097770592,20140807,00,00,0006679,0,00,0,,,,,0,,
52,123456780,20140808,,44000002,,,,0,,,,,0000,,0,,0006679,"c:/users/x9ware5/documents/
x9ware/files_Utilities/Test file with 2 checks
exporter_IMAGES/Bundle_000003/Image_000013_amount_10004_front.tif"
50,1,097770592,20140807,00,00,0001865,1,00,0,,,,,0,,
52,123456780,20140808,,44000002,,,,0,,,,,0000,,0,,0001865,"c:/users/x9ware5/documents/
x9ware/files_Utilities/Test file with 2 checks
exporter_IMAGES/Bundle_000003/Image_000015_amount_10004_back.tif"
70,0002,000000020006,000000020006,00004,,
90,000001,00000002,00000000020006,000000004,"File Generator",20140810,
99,000001,00000018,00000002,0000000000020006,,,
```

File ID Modifier XML File

X9Utilities allows your File ID Modifier to be either assigned as an explicit value or assigned indirectly from a File ID Modifier XML File. Some processors require that the File ID Modifier be unique for a given file creation date. This requirement is typically based on their duplicate file prevention logic.

The File ID Modifier XML File can be used as a tracking file for the automated assignment of unique File ID Modifier values within the same processing day. When using this technique, you will get a unique File ID Modifier (values from A through Z) each file creation date. For example, the first would be created with a File ID Modifier of “A”, the second with “B”, the third with “C”, and so on. This is done automatically with no action on your part.

This facility is activated using the assigned File ID Modifier value as follows:

1. If the File ID Modifier is one character in length, then it is accepted and explicitly used.
2. If the File ID Modifier is four characters in length with a value of “auto”, then a tracking file is internally allocated by X9Utilities and will be used to assign sequential File ID Modifiers.
3. If the File ID Modifier is greater than one character in length and is not “auto”, then it must be the fully qualified name of your XML tracking file. This file is used as input to get the late date and File ID Modifier that was assigned and is then updated by the current run. If the specified file name does not exist, then it will be automatically created.
4. Consecutive values are assigned within the same calendar day as follows:
    - “identifierIsNumeric” can be set to a value of true which results in the assigned consecutive alternate values of '1' through '0'.
    - “identifierIsAlpha” can be set to a value of true which results in the assigned consecutive alternate values of 'A' through 'Z'.
    - Otherwise the value is assigned consecutively first from “A” through “Z” and then from “1” through “0”.

A same File ID Modifier XML file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<fileIdModifier>
 <copyright>X9Ware LLC 2012 2013</copyright>
 <product>X9Utilities</product>
 <release>R4.01</release>
 <timestamp>20191120_070349_148</timestamp>
 <fields>
  <fileDate>20141120</fileDate>
  <identifierIsNumeric>false</identifierIsNumeric>false
  <identifierIsAlphac>false</identifierIsAlpha>false
```

```
<fileIdModifier>auto</fileIdModifier>
 </fields>
</fileIdModifier>
```

Supported x9 Configurations

The following x9 configurations are supported by X9Utilities:

Configuration	Usage
x9.dstu-no-field-validations	File level structural validations are applied with minimal field validations.
x9.37	The DSTU x9.37 which was the first x9.37 standard originally defined in 2003 and is the most basic and commonly used standard throughout the industry.
x9.100-180-2006	The x9.100-180-2006 standard which is very infrequently used within the industry due to more drastic changes that were implemented (including the type 52 image record) and the associated complexities that resulted from the technical direction that was taken.
x9.100-187-2008	The x9.100-187-2008 standard which is widely accepted through the industry.
x9.100-187_UCD-2008	The x9.100-187-2008 standard with the associated Universal Companion Document applied. This standard is widely accepted by many financial institutions.
x9.100-187-2013	The x9.100-187-2013 standard which is increasingly accepted through the industry. This standard includes the type 62 credit record and was the first (beyond x9.100-180-2006) to add credit support.
x9.100-187_UCD-2013	The x9.100-187-2013 standard with the associated Universal Companion Document applied. This standard is increasingly accepted by many financial institutions.
x9.CPA_015	The Canadian CPA 015 standard which is based on x9.100-187-2006 and is implemented within Canada.
x9.Frb	The Federal Reserve Bank (FRB) standard which is accepted by a large number of institutions and processors.

Supported x9 Record Types

Each item can contain the following x9 record types. Please refer to the appropriate x9 documentation for the format of these records.

Type	Record Name	Notes
25	CHECK_DETAIL	“Write” can create credits as record type 25 when needed by your financial institution
26	CHECK_ADDENDUM_A	
27	CHECK_ADDENDUM_B	
28	CHECK_ADDENDUM_C	
31	RETURN_DETAIL	
32	RETURN_ADDENDUM_A	
33	RETURN_ADDENDUM_B	
34	RETURN_ADDENDUM_C	
35	RETURN_ADDENDUM_D	
61	DSTU Credit Format	Contains 12 fields per the DSTU credit format
61	Metavante Credit Format	Contains 13 fields per the Metavante credit format
62	CREDIT	
68	User Record	Content is per user defined requirements

X9.37 Data Types

Per x9.37 exchange standards, the following data types may be defined and used within x9 files:

Data Type	Description	Content
A	Alphabetic	The alphabetic characters are the upper-case letters A through Z; the lower-case letters a through z, and the blank (space) character. When lower case letters are used, they are interpreted as having the same meaning as their respective upper-case letters. However, use of lower case characters is discouraged; all alphabetic characters should be presented in upper case.
N	Numeric	The numeric characters are the numbers zero (0) through nine (9).
B	Blanks	The blank character is defined in ASCII with the value 0x20 and in EBCDIC with the value 0x40.
S	Special Characters	Special characters are any printable characters with an ASCII value greater than 0x1F or an EBCDIC hexadecimal value greater than 0x3F that are neither alphabetic, nor numeric, nor blank. Occurrences of values EBCDIC 0x00 through 0x3F and ASCII 0x00 through 0x1F‘ are not valid.
AN	Alphameric	An alphameric character is any of the alphabetic or numeric characters.
ANS	Alphameric Special	An alphameric/special character is any one of the alphabetic, numeric, or special characters.
NB	Numeric Blank	A numeric blank character is any one of the numeric characters or the blank character. Blanks shall not be embedded within the numeric digits, but instead can only be used to fill out a field when the numeric value length is shorter than the actual field length. Leading zeros are allowed.
NS	Numeric Special	A numeric/special character is any one of the numeric characters or special characters.
B	Binary	Any binary byte with a value from 0x00 through 0xFF.
NBSM	Numeric Blank Special MICR	A numeric blank special MICR character is any one of the numeric characters, the blank character, the asterisk character, the dash character, or the slash character.

Write

Write is an advanced tool that can be used to easily create new x9 output files. In order to simplify the overall x9 file creation process, Write controls the creation of bundles and their associated trailer records. It calculates and inserts all required totals within the bundle, cash letter, and file control trailer records. All of this is done automatically behind the scene. Writer input is created by your internal applications in a simple CSV format. Images must be provided in TIFF image exchange (x9.100-181) format as typically created by all industry scanners. A parameter file allows you to specify various options including header and cash letter header values which are provided externally in an XML file. Through the use of these header file parameters, write is designed with the goal of being able to create an x9 file for any financial institution. We have incorporated a large number of options which already make this possible for many endpoints and we continue to enhance this process to work towards that goal. This may not always be possible given the complexities that have been implemented by some processors. Please let us know where we can make enhancements which may involve consulting costs subject to level of difficulty and uniqueness of the required solution.

The following information is provided to write:

- **Command line options:** identifies inputs, outputs, and optionally the HeaderXml file to be used to control formatting.

- **HeaderXml:** identifies header, trailer, and overall formatting of the x9 file to be created. There are 100+ parameters within the HeaderXml file which can be used to generate the output x9 file in the format as required by your financial institution. The HeaderXml file is static and typically does not need to be modified once a new implementation is tested and has been moved to production status. The HeaderXml file must be either identified on the command line or on the first row of the input CSV file. See the appendix for a full definition of available HeaderXml file content.

- **Items CSV:** defines the items to be written including amount, sequence number, MICR routing, MICR OnUs, MICR AuxOnUs, EPC, etc. Several formats are supported for the Items CSV file. The most generic and easy to use is the single line “t25” format which has eleven (11) columns. Using that “t25” format allows the items CSV to be generic in nature, with all formatting information required by your financial institution to be provided via the HeaderXml parameter file. See the appendix for a full definition of available HeaderXml file content.

- **Front and Back images**: which are referenced at the item level from your Items CSV and thus externally defined in your file system. These images are highly recommended to be encoded in the TIFF image exchange x9.100-181 standard format (which is the format created by all compliant scanners). Images can be provided in other formats (JPG, BMP, GIF, or PNG) but there are potential performance implications when doing so given the processing time associated within image conversion.

Addendum records can be included on your CSV file and then incorporated into the x9 output file. Your input can include type 26 (BOFD) primary endorsement records or can include type 28 secondary endorsement records. This similarly applies to the endorsement record types for returned item files.

Command line options

Switch	Description
-xml:	Fully qualified headerXml file to be used for this run. The headerXml file name should be enclosed in quotes. Example usage would be -xml:"c:/users/userid/foldename/headerXml.xml".
-j	A summary JSON file will be created with a suffix of "_summary.json" in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of "_summary.txt" in the same folder as the output x9 file.
-x	A summary XML file will be created with a suffix of "_summary.xml" in the same folder as the output x9 file.
-l	Will list all csv lines to the system log.
-dpi:	Assigns the output image dpi used to draw images, with a default of 240. Assigned values must be either 200 or 240.
-imageRepairEnabled	Enables automated image repair; this function is disabled by default.
-imageResizeEnabled	Enables both automated image repair including automated image resize; these functions are disabled by default.
-enp	The "end" statement is not provided on the csv input file. Setting this flag will eliminate an abort when the "end" statement is not present. This flag should be used cautiously, since the purpose of the "end" statement is to ensure that all items have been read and processed.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -write <input.csv>

reads <input.csv> and creates and <input.x937> within the designated folder. The HeaderXml must be defined as the first row within the CSV items file. Note that we suggest that you do not use this alternative, but instead take advantage of the flexibility that is provided when you define HeaderXml on the command line. See the appendix for a full definition of available HeaderXml file content.

x9util -write <input.csv> <HeaderXml> <output.x937>

reads <input.csv> and creates <output.x937> at the fully qualified folder location. The HeaderXml file is explicitly defined on the command line. See the appendix for a full definition of available HeaderXml file content.

x9util -x -write <input.csv> <HeaderXml> <output.x937>

reads <input.csv> and creates <output.x937> and <output.xml> at the fully qualified folder location. The HeaderXml file is explicitly defined on the command line. A summary XML file will be created. See the appendix for a full definition of available HeaderXml file content.

x9util -j -write <input.csv> <HeaderXml> <output.x937>

reads <input.csv> and creates <output.x937>, <output.xml>, and <output.txt> at the fully qualified folder location. The HeaderXml file is explicitly defined on the command line. A summary JSON file will be created. See the appendix for a full definition of available HeaderXml file content.

X9 Configuration Reference

For most X9Utilities commands, the x9 configuration is provided via the command line using the "-config" switch. However, -write instead obtains this parameter from the headerXml file. Because of that, it is not appropriate to provide the x9 configuration via the -config switch.

Automated Image Repair

Write has the ability to automatically repair and/or resize images as part of loading images and constructing the type 52 image records. Although this functionality exists, it is disabled by default with our recommendation that it should not be utilized. Why is that? This is because your application environment is the "system of record" for the images that are being created and distributed. These images are the legal facsimile for these items. The receiving financial institutions will be storing these images in their image archives for 7+ years for their internal use and reference. Because of this, the creation of images is one of the most critical aspects of your x9.37 image exchange application, where these images must be compliant with the x9.100-181 standard. You application should not defer these responsibilities to image conversion code that is embedded in our "-write" function. There are several reasons for this:

- You will not have a copy of the repaired image (it will only be in the x9.37 image file).
- You will have no positive indication as to whether the attempt to repair the image has been successful. If the repair is unsuccessful, then there will be no image present. Even if the repair is successful, you will have no control over the final quality of that image.
- The image may be resized, which means that it will no longer represent the physical size of the original item.

- Finally, image repair is an expensive operation and will dramatically slow the creation of your output x9.37 file. When the images are repaired at this point in the process, it is being done within the critical timeline path. It is much better to do it earlier.

If you are looking for tools, one option is: http://www.unisoftimaging.com/ . This product is able to do apply good thresholding conversions to very difficult images, include Postal Money Orders.

MICR Line

The E-13B MICR characters consist of ten specially designed numbers (0 through 9) and four special symbols (Transit, Amount, On-Us, and Dash). These characters and their usage on the MICR line are described in detail in the "MICR Line" topic. Please refer to that section wihtin this manual for more information.

HeaderXml Reference

The headers format can be provided on the command line (using the "xml" switch) or as the first line within the items CSV file. There is no preference to these alternatives. Both are supported, so you should choose the approach which works best for your application. Providing the HeaderXml reference on the command line can be helpful since it makes the items file more generic and independent of the target customer. Providing the HeaderXml reference with the items file allows the csv to be a more complete package that defines the items as well as the wrapper rules. See the appendix for a full definition of available HeaderXml file content.

Items

Forward presentment items are presented in one of several manners:

Line Type	Field Count	Usage	Columns
t25	11	This format of the "t25" line allows all common type 25 fields to be specified explicitly as a single row. Virtually all user applications will find that they can use this t25 format to represent their items completely and as the only line types within their CSV file. It is our recommended format. This has the extreme advantage that each item will be represented by a single line (row) within the CSV file, which is convenient since the CSV will then be very easy to view in either a text editor or in a spreadsheet tool. The t25 fields are designed to include all of the fields that are typically variable at the item level. All information that is typically constant (record	"t25" (quotes not needed) Amount Item Sequence Number Routing (includes check digit) On-Us Auxiliary On-Us External Processing Code Image Creator Routing Image Creator Date Front-Image file name Back-Image file name

Line Type	Field Count	Usage	Columns
		level indicators, etc) are assigned from the HeaderXml definition. Front and back image file names must be fully qualified and represent the location of those TIFF image files within your file system. These files names must typically be enclosed within double quote marks since they may contain embedded spaces. Image Creator Routing and Image Creator Date can be specified in this format at the item level, but more typically will be omitted and instead deferred to the HeaderXml definition. Image Creator Routing identifies the entity who has physically captured the image for this item (it is not the MICR routing, which represents the payor institution). The Image Creator Date indicates that the date that this image was captured (converted from paper to image). Typically, the combination of image creator routing, image creator date, and item sequence number represent a unique key that could be used to identify and retrieve an item image from an image archive. This “t25” format can optionally insert BOFD (type 26) and secondary (type 28) endorsements that are created from the HeaderXml parameters, when they are required by your financial institution. As an alternative to that simplified approach, you can also explicitly provide the addenda records as individual rows within your CSV file, after each record. You would only want to do that when the endorsement record content is not constant across all items but is instead variable on an item by item basis. In that situation, the addendum record(s) would follow the “t25”; the 7-field format of the “t25” format must be used, with images on their own subsequent lines.	
t25	12	This format of the “t25” line is exactly at the above but includes a batch profile column which is the 12th column within each item row. Batch profiles are used to allow multiple credits to be created in a single x9 processing run. Use of this	“t25” (quotes not needed) Amount Item Sequence Number Routing (includes check digit) On-Us Auxiliary On-Us

Line Type	Field Count	Usage	Columns
		facility creates a more complex x9 file, where items are grouped by depositor, and where each deposit (a credit offset by a series of checks) represents the items for a specific depositor relationship. Our use of batch profiles allows your item CSV file to remain generic in nature, where you build a profile which identifies specific information for each depositor relationship (such as credit routing, credit account number, etc). Individual batch profiles must exist with the same folder where your HeaderXml definition resides. You then create a batch profile for each customer that specifies their detailed credit information. For example, you might create batch profiles named customer1, customer2, customer3, etc. Each of these batch profiles then contain detailed information on the credit to be manufactured for a specific customer. The following fields can be optionally specified via the batch profile definition. In reality, typical usage would only specify routing, OnUs, and AuxOnUs, with the other values being assigned directly from HeaderXml and thus applied to all credits. The full list of available fields are as follows: • creditAccountName • creditStructure • creditPayorBankRouting • creditMicrOnUs • creditMicrAuxOnUs • creditItemSequenceNumber • creditRecordUsageIndicator • creditDocumentationTypeIndicator • creditTypeOfAccount • creditSourceOfWork • creditDebitCreditIndicator • creditReturnAcceptanceIndicator • creditMicrValidIndicator • creditBofdIndicator • creditAddendumCount • creditCorrectionIndicator • creditArchiveTypeIndicator	External Processing Code Image Creator Routing Image Creator Date Front-Image file name Back-Image file name **batchProfileFileName** Use of our profiling functionality is an advanced topic. Batch profiles are text (property) files that must be defined within the same folder as where the HeaderXml file is located. The profile name should include the file extension. For example, “AcmeConstruction.txt” would be a properly formatted batch profile file name. The batch profile file name can include blanks; in that case they would have to be quoted. A fully qualified batch profile file name is constructed from the HeaderXml folder name and the batch profile file name. Please contact X9Ware for more information.

Line Type	Field Count	Usage	Columns
		A typical HeaderXml redirection would look like: <creditPayorBankRouting>//creditPayorBankRouting</creditPayorBankRouting>/ <creditMicrOnUs>//creditMicrOnUs</creditMicrOnUs> <creditMicrAuxOnUs>//creditMicrAuxOnUs</creditMicrAuxOnUs> And then a customer batch profile would look like: creditAccountName=Test Account creditPayorBankRouting=555555550 creditMicrOnUs=112233445566/ creditMicrAuxOnUs=	
t25	7	This format of the “t25” line allows certain type 25 values to be specified explicitly (amount, item sequence number, and the MICR line as individual parsed fields) while the various indicators are defaulted from the HeaderXml parameter file. Use of this “t25” format can simplify your CSV definition but requires that the same indicator values are assigned to all items. This format requires that “image” lines follow the “t25” line to define the front and back images to be inserted into the item.	“t25” (quotes not needed) Amount Item Sequence Number Routing (includes check digit) On-Us Auxiliary On-Us External Processing Code
t25	4	This format of the “t25” line allows certain type 25 values to be specified explicitly (amount, item sequence number, and the MICR line as a single scan line which will be subsequently parsed) while the various indicators are defaulted from the HeaderXml parameter file. You can use this option when want to defer parsing of the MICR line to the writer. Use of this “t25” format requires that the same indicator values are assigned to all items. This format requires that “image” lines follow the “t25” line to define the front and back images to be inserted into the item. In order to parse the MICR	“t25” (quotes not needed) Amount Item Sequence Number MICR scan line See the appendix for a full definition of available HeaderXml file content.

Line Type	Field Count	Usage	Columns
		line, values must be provided via HeaderXml for the various separator characters: micrTransitSymbol which defaults to 'A'; micrAmountSymbol which defaults to 'B'; micrOnUsSymbol which defaults to 'C'; micrDashSymbol which defaults to 'D'.	

Type 25 records would normally be provided using the “t25” line. It has the benefits that certain fields (such as the documentation type indicator) will be assigned from HeaderXml and will not have to be hardwired into your application program. This not only simplifies things, but also allows your application to more easily work with multiple endpoints when they require these various indicators to be assigned differently. However, in some situations, you can need to take full and explicit control of the type 25 record. In that case, this line layout can be used:

Line Type	Field Count	Usage	Columns
25	15	The record type 25 definition allows you to provide a complete list of all of the values that are defined for the check detail record. Use of this format give you total control over the data that is populated for each item. This method is required when the various record type 25 indicators are not constant and can change from item to item.	Record Type Auxiliary On-Us External Processing Code Routing (8 digits) Routing Check Digit On-Us Item Amount Item Sequence Number Documentation Type Indicator Return Acceptance Indicator MICR Valid Indicator BOFD Indicator Addendum Count Correction Indicator Archive Type Indicator

Type 26 records would normally be generated using the facilities provided via HeaderXml definition. However, this may be inadequate since that approach assigns the same basic type 26 record to all items. There are times when certain fields must be dynamically populated for each item (eg, deposit account number or payee name). In that case, this line layout can be used:

Line Type	Field Count	Usage	Columns
t26	5	The “t26” line can provide an appropriate type 26 BOFD field values tor the current item. The “t26” would immediately following the “t25”.	“t26” (quotes not needed) Return routing Deposit account number Deposit branch Payee name

Return items can only be presented in the type 31 return record format. This is because the various indicators (and especially the return reason) will most probably vary from item to item, hence the format must provide the flexibility to change these values. The CSV layout is as follows:

Line Type	Field Count	Usage	Columns
31	14	The record type 31 definition allows you to provide a complete list of all of the values that are defined for the check detail record. This gives you total control over the data that is populated for each item.	Record Type Routing Number Routing Number Check Digit On-Us Return Record Item Amount Return Reason Addendum Count Documentation Type Indicator Forward Bundle Date ECE Institution Item Sequence Number External Processing Code Return Notification Indicator Return Archive Type Indicator Reserved

Credits

Credits can become a complex topic, since credit definitions are essentially extensions to many of the x9 standards. X9Utilities supports all common credit formats and can be further enhanced to support additional formats if needed by your organization. An XML field defines which credit format is to be generated, which allows the data to be generic. An XML field is also used to define the relative location of where the credit is to be inserted within the x9 file that is being created.

Credits can be inserted into the generated x9 file using one of several techniques, subject to your design and approach to file creation. Options are as follows:

Approach	Usage	Number of Credits?	Defined In	Examples
Auto insert	A single credit can be inserted with the amount calculated	One	XML only	This approach does not require any account level information in the

Approach	Usage	Number of Credits?	Defined In	Examples
	from the offsetting debits. Various xml fields are used to populate all required fields for the chosen credit format. Inserting the credit automatically greatly simplifies the generation of the credit, since the content is defined by xml parameters and the credit amount is calculated automatically from the items. When credits are inserted automatically, they can then have images optionally attached when needed. The image can be drawn dynamically using the creditImageDrawFront and creditImageDrawBack parameters. They can alternatively be defined externally (perhaps as a generic proxy that is always used) using the creditImageProxyFront and creditImageProxyBack parameters.			CSV file. The credit amount is calculated automatically from the offsetting items. All parameters needed to create the credit are defined in HeaderXml. This does mean that the created x9.37 file will be for a single depositor. Here is an example of the required XML parameters: creditFormat=t25 creditRecordLocation=a20 creditInsertedAutomatically=true creditPayorBankRouting=123456780 creditMicrOnUs>1122334455/005 creditItemSequenceNumber=auto creditDocumentationTypeIndicator=G creditBofdIndicator=U creditImageDrawFront=true creditImageDrawBack=true creditImageDrawMicrLine=true creditAddToItemCount=true creditAddToTotalAmount=false creditAddToImageCount=true
Explicitly defined credit in generic format	A single credit or multi-credit file can be created using credit rows which are defined within the items csv file. The credit line includes all commonly used credit fields. All other credit fields are populated from xml. The csv file should be constructed as a series of deposits, where each deposit is a credit offset by checks. The credit insertion point should be defined as “any” to allow it	Multiple	CSV and XML	credit,351420,44000001,555555550,”12345678/”,330000444, There are 7 columns with content as follows: “credit” amount sequence number routing OnUs AuxOnUs EPC

Approach	Usage	Number of Credits?	Defined In	Examples
	to be written in the order as provided on the items csv file.			
Explicit definition using specific record type and fields	Credits can be provided on an explicit basis as individual rows within the items csv file. These credits can be 61 or 62 record types, but can also be type 25s that are masquerading as credits. The approach of using type 25s is typically used by banks to construct a deposit file that can be accepted by their capture system. It requires some proprietary technique to identify the credits, which is typically a transaction code that is embedded within the MICR OnUs field.	Multiple	CSV only	The number of CSV columns is variable number on the credit format that is used: 61,2,44000001,,555555550,”12345678/”, 44000001,”G”,0,,
Simple single credit	A simple credit can be provided within the items csv file using an explicitly provided amount. This format is typically used when there is only a single credit and when you want to provide the dollar amount (you do not want it to be automatically calculated). All required credit fields are populated from xml along with the credit format and insertion point.	One	CSV and XML	credit,351420 There are 2 columns with content as follows: “credit” amount

Excessive Field Sizes

Write will set an exit status of 3 if there are any fields that have been assigned values that exceed their maximum defined size, per the x9.37 standards. An example would be an attempt to assign MICR OnUs with a 22 character value, or MICR AuxOnUs a 17 character value. In these situations, the input values will be truncated and the error condition included in the system log, along with the assigned exit status.

Credit Types

X9Utilities supports the following types of credits (please refer to the Addenda definitions of these various record layouts):

Credit Type	X9Ware Format Number	Introduced by X9.37 Specification	Record Length	Number of Fields	Description
61 “Metavante“	001	N / A	80	13	This credit layout was defined very early after the 2003 introduction of the X9.37 standard and is commonly used by many financial institutions given is acceptance throughout the industry.
61 “DSTU“	002	N / A	80	12	Our reference to this credit format as “DSTU” is a misnomer, only because it was never officially accepted as part of the DSTU standard. However, it was defined at virtually the same time as the 2003 DSTU standard itself, and used by many financial institutions. Our opinion is that it is not accepted as widely as the Metavante layout.
61 “X9.100-180“	003	x9.100.180-2006	82	12	This credit layout was introduced with the x9.100-180 standard in 2006. Note that the record length is 82 and not 80, so it is very different from most other credit layouts. It is used infrequently given that the x9.100-180 standard was generally rejected by most banks and processors.
62	000	x9.100-187-2013	100	12	This credit layout was introduced with the x9.100-187-2013 standard to address the lack of a credit standard. Note that the record length is 100 and not 80, so it is very different from most other credit layouts. This credit format has the advantage that x9.100-187-2013 explicitly

Credit Type	X9Ware Format Number	Introduced by X9.37 Specification	Record Length	Number of Fields	Description
					defines the impact of credits on trailer record content.

Credit Images

Front and back images can be optionally attached to inserted credits. Many financial institutions do not require credit images to be attached to type 61 or type 62 credits. However, images are often needed when the credit is inserted as a type 25 record.

Several techniques can be used to insert the credit images. All of these allow either one or two images to be attached to the credit. When one image is attached, it will always be assumed as the front image. When two images are attached, they will be the front image followed by the back image. Alternatives are as follows:

Approach	Applicability	Description
Proxy Images	Used when credits are inserted automatically.	Proxy images can be inserted using the creditImageProxyFront and creditImageProxyBack parameters. These are external images will be inserted exactly as they are defined.
Drawn Images	Used when credits are inserted automatically.	Dynamically drawn images can be inserted using the creditImageDrawFront and creditImageDrawBack parameters. The drawn front image is a very basic deposit slip that can contain an optional MICR line and will then include the following information: bank name, date, originator name, account name, account number, credit amount, and item count.
Inserted Images	Used when credits are included in the csv file (as “61”, “62”, “credit”, or “25” record types). This approach must be used when the output x9 file will contain multiple credits, and hence they cannot be inserted automatically by the writer.	Item level images can be inserted using the inline “imageFolder” and “image” directives, which immediately follow each csv credit record.

Custom Type 61 Credit Formats

Custom credits can be created when the standard type 61 formats are determined to be insufficient. The need for this procedure would be considered extremely unusual. We instead request that you

contact X9Ware to get any required type 61 credit formats added to our standard x9 rules. If absolutely needed, a procedure to add an unusual type 61 credit format would be as follows:

- The approach is to create a new x9rules definition, which will have the required type 61 credit format. This modified rules definition will replace on of our standard definitions.
- Select the x9rules definition to be used. These base xml documents are stored in / rules / x9rules /. For example, this might be xml document x9rules_x9.100-187.xml.
- Backup the selected xml document to a safe location.
- Using an xml editor, locate the type 61 records that are present. Type 61 record format zero (0) will be retained. Remove all credit formats other than DSTU (format type 2).
- The result is that there is now only one credit format in this x9rules set (other than the generic format zero).
- Now apply updates to this type 61 format 002 definition to mimic the required fields. There are several key requirements during these changes. First is that the field names must be retained, allowing them to be logically identified. Both field lengths and the actual order of the fields can be changed. However, the final number of fields must still be twelve (12) when you are completed. Depending on your requirements, this may imply that several fields be combined into a reserved title. You will need to be a bit creative to ensure that the field count requirement is met.
- Double check that all of your xml field numbers are sequential (1, 2, 3, etc) and that the positions and lengths are all correct.
- Once these updates are completed, use an xml editor to ensure that xml tags are still valid.
- Your new x9rules with your custom type 61 credit definition is now ready for testing.

Item Images

Image information is provided within the CSV file using several unique line types.

Line Type	Column	Content	Description
Image folder definition	1	“imageFolder”	The image folder line is optional and can be used to provide the root directory of your image folder. This facility is not compatible with batch profiles. You would typically specify parameter a single time within your csv file (on the second input line number and immediately after the header parameter) but can instead be repeated whenever the image folder changes (for example, at the bundle level). This image folder is then be used to construct the item level image names.
	2	Image folder file name	The folder name should be specified as a quoted string using forward slash separators. Backslashes are not recommended since they are part of an ESCAPE character sequence and can be unexpectedly ignored. The folder name should not end with the separator character. We highly

Line Type	Column	Content	Description
			recommend enclosing the input folder within quote " marks, which are mandatory when the folder name contain embedded blanks.
Image file definition	1	"image" or "image2"	The "image" line defines a front or back image to be attached to the current item. Two "image" lines would typically always be defined for each item, per image exchange rules. The first image line represents the front side image and the second image line represents the back-side image. These two lines must follow their associated type 25, 31, 61, or 62 records; there may be user defined type 26 or type 28 records that precede these image lines. As an alternative, the "image2" line defines a multi-page tiff image that contains the front and back images as two pages, within a single tiff image. When "image2" is used, the multi-page image will be internally separated into two separate images that are then redrawn into two images, using x9.100-181 standards, The original tiff tags from the multi-page image are discarded and new tiff tags are assigned. If a paid stamp is being applied to an image, then it must appear before its associated image line.
	2	Image file name or base64 image string	The image file name should be specified as a quoted string using forward slash file separators, enclosed within quote marks. Please note that backward slash should not be used since it is processed as part of an ESCAPE character sequence and will be ignored. The image can also be provided as a hexadecimal base64 string. In this case, the image descriptor is provided using a prefix of base64=, which identifies the image as a base64 string (and not a file name). This approach allows the CSV to be completely self defined, since it will not contain file name references. This is advantageous when the CSV file is being created by a pre-processor and then run through X9Utilities at a later point in time.
	3	Image creator routing number	Identifies the image creator routing number and is populated in field 50.03. If this value is not specified, it will first be defaulted first from xml field <itemImageCreatorRouting> and then secondly from field <cashLetterEceInstitutionRouting>.

Line Type	Column	Content	Description
	4	Image creator date	Identifies the image creator date and is populated in field 50.04. If this value is not specified, it is defaulted from xml field <createDate>.

If the provided image is in TIFF format, then every attempt will be made to use that image exactly as presented in the created type 52 record. The TIFF image will be loaded and inspected. If it is determined to be a valid TIFF image that meets x9.100-181 image exchange standards, then the TIFF byte array will be inserted as provided into the type 52 image record. If a TIFF image is presented, we recommend that you ensure that it is already in x9.100-181 compliant format.

By default, image re-size and re-scale are not enabled within X9Utilities. These actions are purposefully disabled (by default) for several reasons. First is that these operations are CPU intensive and not recommended for any high-volume processing situation. It is far better to do this elsewhere, outside of your file delivery path, than doing it within the actual x9 file creation process where the results cannot be reviewed. Second is that image conversion is not an exact science. This conversion can provide unexpected results, and especially when there are color or gray scale translations needed based on how the image was captured. As a result, these capabilities will have to be enabled on the command line to become effective.

Images can be presented in other formats (eg, PNG, JPG, or GIF) but this is not recommended for these very same reasons. If the image must be converted to TIFF, there is also an attempt to determine the original DPI and then re-scale the image to x9.100-181 exchange standards as needed. The ability to determine the original DPI is dependent on the encoding image format and the embedded meta-data.

Paid Stamp

Virtual paid stamps are most typically applied by the scanner during the capture process. They can also be applied by many financial institutions to the file after it has been transmitted to them, as part of their image capture system. If both of these options are not available to you, then the virtual paid stamp can be applied to the back side TIFF image by X9Utilities during the x9.37 file creation process. This facility modifies incoming images by dynamically drawing a paid stamp using the provided text and font sizes. Be aware that drawing paid stamps will add substantially to run time, since every back side image must be redrawn. The new TIFF image will be written at the same DPI as the input image. The created image will comply with the x9.100-181 standard, but the actual tiff tags which are present may be different than those present on the original image.

The incoming TIFF image must be in valid format to allow it to be parsed and updated. If a failure occurs during this process, there will be associated errors in the system log, and the original image will be written (which would not include the paid stamp).

The most typical usage would be to provide a single paid stamp as the first line within your csv file. This paid stamp definition will then be applied to all items. Alternatively, there can be multiple paid stamp lines within the csv file. In this situation, the paid stamp will be applied to

subsequent items until another paid stamp is encountered. Use of multiple paid stamps is not compatible with batch profiles, since csv lines would be reordered by credit account.

The paid stamp definition consists of four parts:

- The title, which will is drawn on the paid stamp borders.
- A variable number of heading lines, which will be drawn in a larger font.
- An empty (blank) string, which separates the heading lines from the text lines.
- A variable number of text lines, which will be drawn in a smaller font.

Text will be drawn using the arial font. Font size can be set separately for the title, heading, and text areas. Each font size can have an optional trailing “b” to indicate that it should be bold.

Paid stamp example:

```
paidStamp,11,10,10,1.800,"Company Name","For Deposit Only","Bank
Name","Other","Instructions","As Needed"
```

Paid stamp example that draws as a filled box:

```
paidStamp,11,10,10,1.800,"box", "fill", "Company Name","For Deposit Only","Bank
Name","Other","Instructions","As Needed"
```

Paid stamp example that draws as a filled area but does not have an enclosing box:

```
paidStamp,11,10,10,1.800,"nobox", "fill", "Company Name","For Deposit Only","Bank
Name","Other","Instructions","As Needed"
```

Line Type	Field Number	Usage	Usage
Paid stamp definition	1	“paidStamp” to be applied. When using batch profiles with creditStructure with bundledCredits, it is important the paid stamp be repeated for every item. This is because the csv input must be grouped and reordered.	Paid stamps are optional and are applied to one ore more items.
Title font size	2	Font size with optional trailing “b” (to indicate bold).	Example is “12”.
Heading font size	3	Font size with optional trailing “b” (to indicate bold).	Example is “12”.
Text font size	4	Font size with optional trailing “b” (to indicate bold).	Example is “10”.
Location	5	Location of paid stamp from the right edge	Suggestion is to use 1.80

Line Type	Field Number	Usage	Usage
		of the back side image.	inches.
Box	variable	Optional string “box” or “nobox” which indicates if a rectangular box is to be drawn around the paid stamp.	Default is “box”.
Fill	Variable	Optional string “fill” or “nofill” which indicates if the paid stamp area is to be filled prior to drawing. When the paid stamp is filled, any information in that area will be obliterated with white pixels prior to the drawing of the paid stamp lines.	Default is “nofill”.
Title text	variable	Defines the title which is drawn just inside of the top and bottom border lines.	Typically identifies the issuer.
Heading text	variable	One or more heading lines.	Typically includes “For Deposit Only”.
Separator	variable	Empty (blank) line which is a separator between the heading lines and the text lines.	
Additional text	variable	One or more text lines.	Additional text lines to describe the item such as payment instructions, store location, etc,

Batch Profiles

The use of batch profiles allows the csv input stream to be utilized in many interesting ways. Batch profiles should be considered when you are using the writer to create multiple deposits to varying accounts.

When batch profiles are not being used, the csv input is processed in exactly the same order as it is presented to the writer. However, when batch profiles are being used, the csv input is reorganized (grouped) based on the assigned batch profile name. For example, suppose you have 100 items that are being deposited to 4 different accounts. Batch profiles allows the 100 items to be reorganized by depositor into 4 separate deposits. All of this is done by the writer, where your responsibility is to assign the correct batch profile name to each item.

There are many benefits to using batch profiles, since the writer will:

- Reorganize the input stream into individual deposits using the assigned profile name.
- Calculate the amount for each deposit from the associated (attached) items.
- Construct a deposit ticket using the appropriate MICR fields for the specific depositor.
- Draw a unique deposit ticket for each credit using their account name.

All of this can be done without using batch profiles. However, it is much more difficult since your application must sort the csv file by depositor, calculate the deposit amounts, and insert “credit” lines into the csv stream. This is possible but obviously more complex.

The easiest way to utilize batch profiles is to incorporate them into your “t25” item line. This works well when creating files using the “t25” format where each item (in its totality) is represented by a single row. However, other more complex writer implementations may require that the batch profile be defined independently. Examples of those more complex flows might be:

Possible flow:
batchProfile
one or more t25’s that include image columns
batchProfile
one or more t25’s that include image columns
end

Possible flow:
batchProfile
paidStamp
one or more t25’s that include image columns
batchProfile
paidStamp
one or more t25’s that include image columns
end

Possible flow:
batchProfile
25 or t25
paidStamp
image or image2
[repeat item groups as necessary]
end

The format of the standalone “batchProfile” line is as follows:

Line Type	Field Count	Usage	Columns
batchProfile	2	Specify the batch profile to be assigned to the subsequent csv lines. This assignment continues until the next batch profile line is encountered. Batch profiles are text (property) files that must be defined within the same folder as where the HeaderXml file is located. The profile name should include the file extension. For example, “AcmeConstruction.txt” would be a properly formatted batch profile file name.	“batchProfile” (quotes not needed) batch profile file name

Line Type	Field Count	Usage	Columns
		The batch profile file name can include blanks; in that case they would have to be quoted. A fully qualified batch profile file name is constructed from the HeaderXml folder name and the batch profile file name. Please contact X9Ware for more information.	

New Bundle Statement

The “newBundle” line is used to indicate that the next item should begin in a new bundle regardless of the current bundle item count. This line can appear anywhere in your CSV input file and typically might precede a credit when you specifically want that credit to begin in a new bundle.

System Log Correlation

The “logger” facility is optional and allows user defined text to be written directly to the system log, for correlation of the system log back to application specific events. In order to use these facilities, it may be useful to also redirect the system log to an alternative folder location, using the “-log” command line switch. User text can be inserted into the system log in several ways. First is by using either the “-logger:” command line switch, for example: -logger:”run 513387”. Second is by including logging lines within the incoming csv file, or example: logger,”deposit 49923814”.

END Statement

The “end” line must be the last line on your input file and logically indicates end of file. The “end” line is mandatory and must always be present. Write will abort if the end line is not present or is not the last line within your input file.

Sample Items File

```
t25,10002,44000001,087770706,"29602722/5526",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
t25,10004,44000002,097770592,"60333044/5587",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
t25,10006,44000003,077770392,"29343913/5178",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
end
```

Sample Items File with a Credit

When you have needed a single credit to offset checks within a deposit, you would normally define that in the HeaderXml file. That definition allows you to specify the type of credit to be inserted, where the credit should be positioned within the file, how the credit impacts trailer record counts and amounts, and also allows you to indicate that a deposit slip should be drawn and attached to the credit in lieu of captured images. All of these advanced HeaderXml capabilities will typically mean that you do not need to include the credit on your items CSV file but would instead just let

the writer create the credit for you. However, you can also take full manual control of the credit and insert it into the file at a position of your choice, and with specific information and attached images. This also allows you to optionally specify the image creator routing and capture date on the created image records. You would do this as follows:

```
credit,20006,44000000,555555550,000000029292/,,
image,"c:pathToFrontImage"
image,"c:pathToBackImage"
t25,10002,44000001,087770706,"29602722/5526",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
t25,10004,44000002,077770392,"29343913/5178",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
end
```

Sample Items File with User Defined 26/28 Records

Sometimes fields within the 26/28 addenda records attached to each item are not constant, but instead must vary from item to item. In this situation, you cannot define your type26/28 records within HeaderXml, since that would attach the same constant data.

When this more complex scenario is needed, you will have to take control of the type 26 (and possible) 28 records that are attached to each item. This approach would allow you to explicitly define the type 26/28 records to be written for each items, and even to have multiple type 26 BOFD records. This is needed to allow the type 26/28 records to be written after the item record and before the type 50/52 image records.

One way to do this is to use the T25 record layout using the 7-field format (without the image names). You can then provide CSV lines for any needed type 26 or type 28 records, followed by the image lines. When using his approach, the image lines can be written with two fields (with just the image file name) or with four fields (which would include the image creator routing and date).

With this implementation, the HeaderXml file will not be completely populated from the bofd, secd, secd2, or secd3 series of fields. These XML fields define type 26/28 addenda records, within the HeaderXml file.

You will have to provide a value for the number of attached addenda records, which is needed for the type 25 record:

```
<itemAddendumCount>1</itemAddendumCount>
```

If the number of type 26/28 addenda records will vary by item, then you will need to use a 25 line that fully defines your type 25 item record (instead of the t25 alternative).

These type 26/28 records can be inserted using explicit field values. In order to simply things a bit, we also provide a T26 line that allows you to provide key type 26 fields via the CSV file while still deferring basic indicators (etc) to Headerml. An example CSV file that uses a 7-field T25 and a unique T26 addenda record for each item would be as follows:

```
t25,10002,44000001,087770706,"29602722/5526",,
t26,123456780,12345,001,"Mary Smith"
```

```
image,"C:/Users/X9Ware5/Documents/x9_assist/files_Utilities/Test file with 2
checks_IMAGES/Bundle_000003/Image_000004_amount_10002_front.tif"
image,"C:/Users/X9Ware5/Documents/x9_assist/files_Utilities/Test file with 2
checks_IMAGES/Bundle_000003/Image_000004_amount_10002_back.tif"
t25,10004,44000002,097770592,"60666044/5587",,
t26,123456780,12345,001,"John Smith"
image,"C:/Users/X9Ware5/Documents/x9_assist/files_Utilities/Test file with 2
checks_IMAGES/Bundle_000003/Image_000010_amount_10004_front.tif"
image,"C:/Users/X9Ware5/Documents/x9_assist/files_Utilities/Test file with 2
checks_IMAGES/Bundle_000003/Image_000010_amount_10004_back.tif"
end
```

Drawn Images and Remotely Created Checks (RCC)

X9Utilities can dynamically draw images which are attached to items. This image drawing process typically applies to one of several flows:

- First is Remotely Created Checks (RCC), where production items are being created.
- Second is Test Center automation, where test items are being created.
- Third is usage to create electronic adjustment files, which can be targeted for production environments.

What is a Remotely Created Check (RCC)?

A check that is not created by the paying bank and that does not bear a signature applied, or purported to be applied, by the person on whose account the check is drawn. The FRB defines a "remotely created check" as a check that is drawn on a customer account at a bank, is created by the payee, and does not bear a signature in the format agreed to by the paying bank and the customer.

RCCs are electronic items where the check images are dynamically created (drawn) and are not captured. Originated RCCs are debits to a specific customer bank account and require authorization of the account owner. RCCs do not bear the signature of account owner but instead include the account holder name within the image itself. The account holder can authorize the creation of the RCC in various means including in-person or telephone. Complete and accurate MICR line information for the customer account must be provided. RCCs are often created as an alternative to ACH by credit card companies or telemarketers. A benefit of RCCs are that they are cleared through the check image clearing networks and RCCs are processed through the check clearing networks and are governed by rules including the Uniform Commercial Code (UCC) and the Expedited Funds Availability Act (Regulation CC). Remotely Created Checks have the potential to be used in situations where ACH may not be available in a specific clearing scenario.

RCCs are similar to their check counterparts. They embody a paper instrument that contains an unconditional written order, instructing a drawee bank (paying bank) to make payment to the order of a designated payee and are processed through the banking system. They serve a useful business purpose in a diverse market, for applications such as Telephone Marketing, Bill Payments, Loan Repayments, Recurring Insurance Payments, and Internet payments.

FED Position as of January 2019 -- Due Diligence Required

Per information from RemoteDepositCapture.com, the Federal Reserve appears to have closed the door on widespread adoption of ECIs, despite widespread industry support. The Fed adopted an amendment to Regulation J in November 2018 (the rule set that governs the collection of checks and other items through Reserve Banks) that effectively bans financial institutions from clearing electronically-created items (ECIs) through the Reserve Bank System which became effective January 1, 2019.

The Fed in explaining its decision said ECIs do not fit Uniform Commercial Code (UCC) and

regulatory definitions of checks since they never exist in paper form. It also said that its decision to ban ECIs from the Reserve Bank System would not stop banks from agreeing to clear ECIs between one another. However, since a significant share of checks clear through the Reserve Bank System – 45 percent on the forward collection side and 68 percent of returns, according to the Fed's data – most experts expect FIs will not encourage business clients to use ECIs in lieu of issuing paper checks.

RCC Items Should be Assigned RPC "6"

As defined in X9 Standard X9.100-160-2014, the EPC is a MICR digit that conveys special information regarding the correct handling or routing of a check or check data to financial institutions and other processors The EPC field is MICR field 6, position 44 on the MICR line, and is located to the immediate left of the Routing number.

A new EPC code of "6" was added to the latest revision of the X9 Standard Magnetic Ink Printing (MICR) X9.100-160-2014 Part 2. Per this standard, the EPC code of "6" shall be assigned when RCC items are created by their originator. This new code, when used appropriately, will create a mechanism for identifying and monitoring these items.

Our RCC Support Leverages "-write" Functionality

The overall process to create to create these image files is very similar to creating any other x9.37 file. The only significant difference is that images are dynamically drawn and are not loaded from externally supplied image folders. When drawing images, the "imageFolder" and "image" directives typically used by "write" are not applicable since images will not be loaded from external TIFF files. Several new directives are added in their place which provide the information that is required to draw the front and back item images.

All other write functionality remains in place and can be leveraged by RCC file creation. This is a tremendous advantage since this common approach allows XML parameters to be provided which control all aspects of the x9.37 file creation process.

Using Image Templates

X9Utilities includes several image templates which can be either used directly or as the basis to build your own customized templates. These standard templates are:

- "rcc1" which is a basic RCC template.
- "rcc2" which is a more complex RCC template that supports the insertion of additional text fields and demonstrates the full range of all capabilities.
- "credit" which is a generic credit document that might be used for internal adjustments.
- "debit" which is a generic credit document that might be used for internal adjustments.
- "testDoc1" which is a test document and is clearly marked as such.

Most usage of our RCC functionality will not want to use our predefined templates but will instead want to define their own custom formats.

- You may have one or more RCC templates
- Each X9Utilies processing run may utilize one more templates within the run
- Templates allow you to position text at the appropriate location based on your design
- Text will be drawn using the font and size of your choosing

Drawing Images

X9Utilities "-write" has the additional capability to dynamically draw front and back images as an alternative to loading them from external image folder(s). Images are drawn from a front image template of your design that is used as a background to construct each individual image. Templates are typically drawn using common tools such as GIMP, Paint, or Photoshop and are saved in a format such as PNG to our internally defined image folder.

Each template consists of a series of internally defined fields which can be placed anywhere within the image based on your template design. These fields can also be drawn with a font and font size of your choosing. These definitions are applied to the "templates.xml" configuration file which is dynamically loaded by the SDK during startup.

Back side images can be created in several manners. This can be a template image of your design, where a variable number endorsement lines are then added. This can alternatively be an image that includes a paid stamp, a variable number of endorsement text lines, and an optional group of "sign here" lines that simulate that area on the back side of a typical check. Finally, for testing environment, the paid stamp can be replaced with a document identifier which can be used to display the internal document type for this item (GL, Cash Ticket, Batch Ticket, etc).

RCC creation has two new CSV line types that are unique to image drawing:

Line Type	Column	Content	Description
Image Creator Definition	1	"imageCreator"	Sets values for image creator date and time.
	2	Image creator routing number	Identifies the image creator routing number and is populated in field 50.03. If this value is not specified, it will first be defaulted first from field "itemImageCreatorRouting" and then secondly from field "cashLetterEceInstitutionRouting".
	3	Image creator date	Identifies the image creator date and is populated in field 50.04. If this value is not specified, it is defaulted from field "createDate".
Draw image definition	1	"drawImages"	Identifies this as a draw images row which must be present for each item.
	2	Front image template name	Identifies the template name to be used for drawing. This template must be defined in the "images" folder and then also must have a definition in "templates.xml" which describes the

Line Type	Column	Content	Description
			coordinates of specific fields within the image.
	3	Back image template name	Identifies the back-side image to be used; this is optional and will be automatically drawn when omitted. When this image is provided, it should have the same physical size as the front image. In addition to the template name, several special values are also available: • "blank" will insert an empty image. • "/identifier" will draw the supplied text on the back side image as document identification. For example "/GL Ticket" will draw and identify the item as GL. • "/$identifier" will draw the supplied text on the back side image as document identification, and also includes a signing block on the far right side of the image. • "/#" will draw a paid endorsement stamp on the back side image. Sign here text can be included by using "/$#". The endorsement stamp includes a series of text fields that are shown within the paid stamp. The paid stamp definition is completely variable and allows title, heading, and text lines to be applied. See the separately documented information on the paid stamp (within this user guide) for more on this format.
	4	Item identifier	Front side image field.
	5	Date written	Front side image field.
	6	Address line 1	Front side image field.
	7	Address line 2	Front side image field.
	8	Address line 3	Front side image field.
	9	Address line 4	Front side image field.
	10	Address line 5	Front side image field.
	11	Payee line 1	Front side image field.
	12	Payee line 2	Front side image field.
	13	Payee line 3	Front side image field.
	14	Memo line	Front side image field.
	15	Bank name	Front side image field.

Line Type	Column	Content	Description
	16	Signature line	Front side image field.
	Lastly	One or more endorsement lines, each presented in subsequent columns.	Back side image. Each text field is prefixed with "/E/" for identification. For example, an example might be "/E/Remote Deposit ISN 8849243". These text lines are inserted into the image and can be used for a variety of application specific purposes. For production files, they can be used as a multi-line endorsement. For test files, they can contain alternate information such as expected results or special instructions.

RCC files will not import images to be attached to each item, but must instead draw them. Most RCC applications will only create debits, but this facility is capable of creating both debits and credits. A n example of a CSV which dynamically creates RCC images is as follows:

```
imageCreator,123456780,20210401
credit,20006,44000001,555555550,"1234567890123/",123456,
drawImages,credit,"/$","123A55001",20140806,"John Doe","1234 Main Circle Dr","Springfield, St
88888-9999",,,"Payee line 1","Payee line 2","Payee line 3","Memo line here","Bank name
here","Signature line here","/E/Endorsement line 1","/E/Endorsement line 2"
t25,10002,44000002,087770706,29602722/5526,,6
drawImages,debit,"/$","123A55001",20140806,"John Doe","1234 Main Circle Dr","Springfield, St
88888-9999",,,"Payee line 1","Payee line 2","Payee line 3","Memo line here","Bank name
here","Signature line here","/E/Endorsement line 1","/E/Endorsement line 2"
t25,10004,44000003,087770706,29602744/5527,,6
drawImages,debit,"/$","123A55002",20140806,"John Doe","23456 Main Circle Dr","Springfield, St
88888-9999",,,"Payee line 1","Payee line 2","Payee line 3","Memo line here","Bank name
here","Signature line here","/E/Endorsement line 1","/E/Endorsement line 2"
end
```

X9Utilities is packaged with the "x9writerRCCtest.bat" batch file which demonstrates the creation of an output RCC file. This sample batch file is as follows:

```
@echo off

: RCC demonstration.
: A sample CSV file defines the items to be created.
: The RCC template can be customized per customer requirements using a paint program.
: Note that this batch script assumes that X9Assist has been installed as the viewer.

set "launchFolder=c:/Program Files (x86)/X9Ware LLC"
set "assistFolder=%launchFolder%/X9Assist Rx.xx"
set "utilitiesFolder=%launchFolder%/X9Utilities Rx.xx"
set "headerXml=%utilitiesFolder%/samples/writer/x9headers.xml"
set "csvInput=%utilitiesFolder%/samples/writer/testFile2ChecksRCC.csv"

: This assignment must be changed to write the output file to the appropriate user folder.

set "x9Output=c:/users/X9Ware5/documents/testFile2ChecksRCC.x9"

: Run x9utilities using the "-write" function to create a new x9 file from an input csv.

cd "%utilitiesFolder%"
```

```
x9util -write -j -l -xml:"%headerXml%" "%csvInput%" "%x9Output%"

: Launch the created x9 file in our viewer as the final step of this demonstration.

cd "%assistFolder%"
x9assist "%x9Output%"

pause

exit /b
```

Draw

Draw reads a CSV file and then writes tiff images, as is provided as a companion tool to -write. The writer function embeds dynamically drawn images directly into the x9.37 file that is being constructed, while draw can be used to create those images and directly save them to external tiff files, instead of inserting them into the type 52 records. Draw can be used to create single page tiff's for the front-back images, but can also be used to create multi-page tiff images, which will be a single tiff image that contains both the front and back within a single image file.

Command line options

Switch	Description
-dpi:	Assigns the output image dpi used to draw images, with a default of 240. Assigned values must be either 200 or 240.
-l	Will list all incoming csv lines to the log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Draw csv line types

Switch	Description
imageFolder	Defines the folder used to load template images that are referenced when drawing images from pre-constructed templates. The folder name is provided as the second parameter, and typically would always be enclosed in quotes since it may contain embedded blanks.
outputFolder	Defines an optional high level folder to be used when writing the output images. This line is not needed when the output image file names are fully qualified. The output folder name can be changed as often as desired. The folder name is provided as the second parameter, and typically would always be enclosed in quotes since it may contain embedded blanks.
paidStamp	Defines a paid stamp to be applied to back-side images. The paid stamp is optional and can appear as often as needed within the csv file, and must appear before the items to which it applies. The paid stamp can appear as often as needed within the csv file, even at the item level.
item	Provided information about each logical item to be drawn. Columns are as follows: • Auxiliary OnUs • External Processing Code (EPC) • Routing • OnUs • Amount • Identifier (any string you desire; for example the item sequence number)

Switch	Description
	• Front image file name • Back image file name. This file name can be defined as “omit”, which means that the back side image will not be created. It can also be defined as “multi”, which means that a multi-page tiff image is to be created that contains the front and back side images within a single tiff file.
drawImages	Provides information for the two images that are to be drawn. Refer to the -write function for documentation and usage.

Command line examples

x9util -draw <input.csv>

reads <input.csv> and creates images per the item level csv lines.

x9util -draw <input.csv> <output.csv>

reads <input.csv> and creates images per the item level csv lines. The output csv is essentially an echo of the input csv, but also contains confirmation of each of the images that are created including their file size.

Translate

Translate creates an output CSV of items, along with their associated exported images and an optional HeaderXml file, all written in a format that is compatible with our Write function. The data is written to a single CSV file that can be browsed to gain insight into how Translate functions, or it can be processed by your applications.

Translate is generally used infrequently since our Export function is typically used to convert an x9 file to an output CSV. Export offers more flexibility with output formats and command line options. Therefore, you would typically always use Export to create output CSV files.

However, Translate can be used when you have an x9 file and want to create test data specifically for input to Write. This is the most common use case for the Translate tool.

Command line switches (parameters) are used to further control the Translate operation. You can use these settings to indicate if you want to include the addendum records in the CSV data that is written and if you want to extract and write the check images. From a design perspective, your application should process only the CSV record types that you are interested in and ignore all others. It is important not to abort if you encounter a CSV record type that is not required by your application, as new CSV record types may be created in the future. Ignoring unneeded record types will make it easier for you to install new releases.

When image extraction is enabled, the embedded image data will be written to an image folder that is selected per the command line. Images will be stored in the exact TIFF format as they appear in the x9 file. TIFF image file names are constructed to allow for easy correlation between an image and its associated type 52 data record. Both front and back images are stored for each item.

The created CSV file includes the name of each image, which provides a direct association between the x9 data and the images. The CSV file can be viewed with tools such as NotePad, NotePad++, MS-Excel or analyzed and processed by your application programs. We generally do not recommend using MS-Excel since it does a very poor job handling large numbers and accommodating files that have a varying number of CSV columns. An alternative is to use the CSV Editor that is provided with X9Assist.

Another usage for Translate is as a tool to convert an x9 file from one format to another. For example, you may want to change the number of items per bundle, insert an offsetting credit into a file that contains only debits, or change from one credit format to another. All of this can be accomplished by using Translate to create a CSV which is then processed through Write, using another HeaderXml file that is defined with the new attributes. In this situation, the “noHeaderXml” and “noCredits” switches should be used to obtain an output CSV that contains only checks (and no credits). These are generally more advanced file translations, so you will need to familiarize yourself with how Translate and Write can be combined to achieve your desired results. You will obviously need to thoroughly test the conversion process.

As part of the X9Utilities installation package, you can take a look at folder : / samples / translate / as an example of a batch file that invokes the combination of translate-and-writer to accomplish the reformatting of an x9.37 file. In this example, a type 61 credit is being inserted into a file that otherwise contained only checks.

Command line options

Switch	Description
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific "-config:" value. For example, the CPA 015 standard can be selected as: "-config:x9.CPA_015".
-a	Includes check addenda records in the output csv file.
-i	Will write images to the output images folder.
-noHeaderXml	Excludes the "headerXml" line from the output CSV file. By default, the headerXml file will be constructed and this line will be included in the output CSV.
-noCredits	Excludes credits (record types 61 and 62) from the output CSV file, along with their immediately attached addenda record. By enabling this parameter, the output CSV file will contain checks only.
-j	A summary JSON file will be created with a suffix of "_summary.json" in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of "_summary.txt" in the same folder as the output CSV file.
-x	A summary XML file will be created with a suffix of "_summary.xml" in the same folder as the output CSV file.
-l	Will list all csv lines to the system log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -translate <input.x937>

translates <input.x937> and creates <input.csv> with associated images written to folder <input_IMAGES>, all within the designated input folder.

x9util -translate <input.x937> <output.csv>

translates <input.x937> and creates <output.csv> with images written to folder <output_IMAGES>, all within the designated output folder.

x9util -translate <input.x937> <output.csv> <imageFolder>

translates <input.x937> and creates <output.csv> with images written to <imageFolder>.

x9util -x -translate <input.x937> <output.csv>

translates <input.x937> and creates <output.csv> and <output.xml> with images written to folder <output_IMAGES>, all within the designated output folder. A summary XML file will be created.

x9util -j -i -translate <input.x937> <output.csv>

translates <input.x937> and creates <output.csv>, <output.xml> , and <output.txt> with images written to folder <output_IMAGES>, all within the designated output folder. A summary JSON file will be created.

Write/Translate Sample CSV files

The following represents a sample CSV file in our recommended format. This is our most commonly used layout and should be utilized whenever possible. It has the following benefits:

- The HeaderXml file is defined externally and would be identified on the writer command line. This approach does not tag the CSV file to a specific HeaderXml file.
- All dates and record indicators are deferred to the HeaderXml file and are not contained within the CSV.
- All information regarding the target financial institution is deferred to the HeaderXml file as well. This is especially useful for a test deck, since it increases overall usability.
- File names are fully qualified and are not referenced as relative to an image folder.

```
t25,10002,44000001,087770706,"29602722/5526",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
t25,10004,44000002,097770592,"60333044/5587",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
t25,10006,44000003,077770392,"29343913/5178",,,,,”c:pathToFrontImage”,“c:pathToBackImage”
end
```

The following represents a sample CSV file where the header record is stored in an external XML file that is referenced from the first line and where the MICR scan lines are provided in their entirety and must be parsed into the auxiliary OnUs, External Processing Code, Routing, and OnUs fields. Providing the MICR line fields in parsed format should always be done when those fields are available within your application. This parsing should only be used when otherwise necessary.

```
headerXml,"c:/users/x9ware5/x9utilities/files/x9headers.xml"
imageFolder,“c:/users/x9ware5/images/testFiles/Bundle1”
t25,"a087770706a29602722c5526c",0000010002,44000001
image,"Image_000013_amount_10002_front.tif"
image,"Image_000015_amount_10002_back.tif"
t25,"a097770592a60333044c5587c",0000010004,44000002
image,"Image_000019_amount_10004_front.tif"
image,"Image_000021_amount_10004_back.tif"
t25,"a077770392a29343913c5178c",0000010006,44000003
image,"Image_000025_amount_10006_front.tif"
image,"Image_000027_amount_10006_back.tif"
end
```

The following represents a sample CSV file where the header record is stored in an external XML file that is referenced from the first line, and type 25 records are fully defined on a field by field basis. This format is most commonly used when the various indicator values on the type 25 record can vary on an item by item basis. When using this CSV format, the front and back images must be specified on attached CSV lines. This format also allows the image creator routing and image creator date to be explicitly (optionally) provided for each item. The format is as follows:

```
headerXml,"c:/users/x9ware5/x9utilities/files/x9headers.xml"
imageFolder,“c:/users/x9ware5/images/testFiles/Bundle1”
25,,,08777070,6,"29602722/5526",0000010002,4400000001,"G",8,1,"Y",00,0,"B"
image,"Image_000001_front.tif"
image,"Image_000001_back.tif"
25,,,09777059,2,"60333044/5587",0000010004,4400000002,"G",8,1,"Y",00,0,"B"
image,"Image_000002_front.tif"
```

```
image,"Image_000002_back.tif"
25,,,07777039,2,"29343913/5178",0000010006,4400000003,"G",8,1,"Y",00,0,"B"
image,"Image_000003_front.tif"
image,"Image_000003_back.tif"
end
```

Export

Export reads an x9 file and then writes the individual x9 data records to an output CSV or XML file. Images can be optionally extracted and individually written to an output folder. The image output folder can be optionally cleared.

Export output can be an exact representation of the x9 file itself, or can be a subset of the records that are present (based on your specific requirements). When images are exported, the image file name will be placed into field 52.19 as proxy for the image itself. This file name is recommended to be fully qualified (using the -i switch) but can also be relative (using the ir switch).

When Export is created using the -xc option, that output can then be used as input to our Import tool. In this manner, export and import are companion tools that complement each other.

Export versus ExportCsv

Export is a very flexible tool that supports a variety of output CSV and XML formats. Export can also operate against either single or multiple input files. All options are accepted via command line switches.

ExportCsv is another alternative, and is different from Export in several important ways. Because of that, you need to consider your requirements and determine which of these tools will best meet your needs. Some of the major differences are:

- ExportCsv allows you do define output columns and their logical order; Export does not.
- ExportCsv allows csv column headers (row one) to be customized; Export does not.
- ExportCsv will only export a single x9 input file; Export will export multiple.
- ExportCsv accepts all parameters through an xml input file; Export takes them from the command line.
- ExportCsv will only export to csv; Export will also export to xml.

Export Formats for X9.37 Output

Export has several output formats that are used to write x9.37 output. These alternatives apply equally to forward presentment and returns. Please use this summary to identify and select the output that is most useful for your specific application.

Switch	Description	When Used	Additional Options
-xf	Output is written in a column format, with one row for each item. The data fields for each item are parsed into fixed columns which make it easy to	This format is used when your application requires the data but you do not want to write the code that is	-dp can be used to include decimal points in the amount fields. -xm can be used to

Switch	Description	When Used	Additional Options
	extract the data that you need. This format is by far the most straight forward to use for your application, when your requirement is generic access to the item data and fields.	necessary to locate data fields across multiple record types. It has the advantage that the standard fixed columns are populated regardless of the x9.37 standard being used.	export on a folder basis (and not just from a single input file). -asis can be used to export the raw data for invalid amount fields; they are otherwise exported as zero.
-xfc	Output is similar to "-xf" but will include a descriptive column names header as the first row in the CSV file. When using this output CSV format, you may also want to limit the record types that are included in the output file. This is an important consideration, since header and trailer records contain different layouts as compared to our standard fixed column format.	This format is used when you want the benefit of fixed fields, where you also want to include a descriptive column header row.	-dp, -xm, and -asis as described above. -rectypes must be used if you want to exclude header and trailer records from your output file. -rectypes:25|26|27|28 should be used to limit record types for forward presentment items. -rectypes:31|32|33|34|35 should be used to limit record types for returned items.
-xc	Output is written on a record by record and field by field basis. The number output rows will exactly match the x9.37 input file. Note that the output format will vary based on the file specification that was used to create the x9.37 file. This is especially true when the input x9 file is in x9.100-180 format.	This format is used when your application requires access to the x9.37 data exactly as it appears on the x9.37 file.	By default, all record types will be written. The -rectypes switch can be used to define only those record types to be included in the output file. -xm can be used to export on a folder basis, which means the entire content of a specified folder, and not just from a single input file.
-xg	Output is written on a group basis, in a fashion that is similar	This format is used when your application	By default, all record types will be written.

Switch	Description	When Used	Additional Options
	to -xc except that the fields for each item are combined. This means that only one row will exist for each item, which will contain all fields from the record types 25, 26, 27, 28 (forward presentment) and 31, 32, 33, 34, 35 (returns).	requires access to the x9.37 data exactly as it appears on the x9.37 file, but you would like the item data as a single row, which eliminates the need to peek forward on record types to see what is present in the file.	The -rectypes switch can be used to define only those record types to be included in the output file. -xm can be used to export on a folder basis (and not just from a single input file).
-xmlf	Output is written in our basic XML flat format.	Usage is dependent on your application requirements.	There are numerous third party XML translators that can be used to convert this output to meet your schema requirements.
-xmlh	Output is written in our more extensive XML hierarchical format.	Usage is dependent on your application requirements. This xml format is generally more flexible than the flat format.	There are numerous third party XML translators that can be used to convert this output to meet your schema requirements.

Command line options

Switch	Description
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific "-config:" value. For example, the CPA 015 standard can be selected as: "-config:x9.CPA_015".
-xc	Export x9 records to csv (which is the default when other export to csv options are otherwise not selected).
-xf	Export items in fixed field format, which can be much easier to incorporate into your application environment, since records and fields for each item have been parsed into specific (fixed) columns.
-xg	Export items in a variable field format (as a single row) which includes the item record followed by all addendum records.
-xm	Exports from multiple input files that are present in a cascading fashion within a provided input folder (this means that the input can contain a mix of files and folders). All input files are read with the output written to a single csv. This function will run for excessively longer times, depending on the number of input files. It takes advantage of

Switch	Description
	multi-threading to reduce runtime. It will take much longer when images are written, given the IO needed to write the images. As a result, consideration should be taken based on your specific application. The -xm switch requires that you identify allowable file extensions via the -exti switch.
-exti	Provides file extensions for multi-file export, where the input is a folder of files and is not a single input file. In that situation, -exti specifies the file extension(s) of the x9.37 files to be selected for export. There can be one or more file extensions which are separated by the pipe ("|") character. For example, -exti:"dat|x937" will select all files from the input folders that have either the "dat" or "x937" file extension. Note that quote marks are needed around this parameter due to the embedded "|" special character.
-xml	Exports to xml (instead of our default which otherwise exports to csv).
-xt	Export tiff tags to csv.
-dp	Inserts decimal points into amount related fields when using the fixed field (-xf) format. Any field which contains "amount" within the field name will have the decimal point automatically inserted. This feature is disabled by default.
-ef	Includes fields which contain blanks data in the xml export.
-ei	Inserts images directly into field 52.19 as base64 strings during xml export.
-i	Exports images to the image folder with absolute file names inserted into 52.19 as proxy for the image data.
-ir	Exports images to the image folder with relative file names inserted into 52.19 as proxy for the image data.
-i64	Exports images in base64-basic format, which are inserted into 52.19.
-i64mime	Exports images in base64-mime format, which are inserted into 52.19.
-tif	Exported images will be in TIF format (this is the default).
-png	Exported images are converted from TIF to PNG format (this can be time consuming).
-jpg	Exported images are converted from TIF to JPG format (this can be time consuming).
-gif	Exported images are converted from TIF to GIF format (this can be time consuming).
-mptiff	Creates and exports a multi-page tiff image to the image folder from the front+back tiff images for each item. Creation of multi-page tiff images is a processor intensive operation due to the associated image encoding and compression activities.
-mpird	Similar to mptiff, this parameter creates and exports a multi-page tiff image to the image folder for the formatted IRD versions of the front+back tiff images for each item. Creation of multi-page tiff images is a processor intensive operation due to the associated image encoding and compression activities.
-ipof	Defines the number of images written per output folder, where the default value of zero creates output image folders that mimic the bundle structure of the current x9.37 file. You can alter this behavior by setting this count to a larger number (eg, 1000) which

Switch	Description
	provides a consistent number of images per output folder, and is especially useful when there are small bundle sizes.
-rectypes	Identifies the specific record types to be included when exporting to csv using either the “xc” or “xf” formats. Record types are identified as a list separated by the pipe (“|”) character. For example, -rectypes:”25|26|27|28” would limit the exported record types to types 25, 26, 27, and 28 on forward presentment files; -rectypes:”31|32|33|34|35” would limit the exported record types to types 31, 32, 33, 34, and 35 on returns files. Fixed field export will always select those record types that are associated with items, regardless of this parameter (for example, you cannot turn off 50-52 record content using this switch when extracting into fixed format). This parameter can be specified as “-rectypes:items” to extract only the item record types, which are 25 thru 52, 61, and 62. It can also be specified as “-rectypes:all” which is also the default assignment. Finally, note that quote marks are often required around this parameter due to the embedded “|” special character.
-skpi:nnn	Indicates a number of seconds that is compared against each individual file creation time and is used to bypass files that are very recently created within the input folder and may be in the process of being transmission. Files that do not meet this minimum skip interval are considered as “in-progress” and will be bypassed. Default value is 60.
-awe	Aborts with an exit status of minus one when the input x9 file is empty. This option is by default disabled, which means that an empty x9 file will produce an empty CSV file with an exit status of zero.
-j	A summary JSON file will be created with a suffix of “_summary.json” in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of “_summary.txt” in the same folder as the output CSV or XML file.
-x	A summary XML file will be created with a suffix of “_summary.xml” in the same folder as the output CSV or XML file.
-l	Will list all x9 records to the system log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -export <input.x937>

reads <input.x937> and creates file <input.csv> within the designated input folder. A default image folder will be created using the input file name.

x9util -export <input.x937> <output.csv>

reads <input.x937> and creates file <output.csv> within the designated output folder. A default image folder will be created using the output file name.

x9util -export <input.x937> <output.csv> <imageFolder>

reads <input.x937> and creates file <output.csv> within the designated output folder with images written to the specified image folder when separately activated using the "-i" or "-ir" switches.

x9util -j -i -export <input.x937> <output.csv>

reads <input.x937> and creates file <output.csv> within the designated output folder. Images are written to the output image folder and those image file names are populated into field 52.19. Summary JSON file will be created.

x9util -x -export <input.x937> <output.csv>

reads <input.x937> and creates file <output.csv> within the designated output folder. A summary XML file will be created.

x9util -xml -i -j -export <input.x937> <output.xml>

reads <input.x937> and creates file <output.xml> within the designated output folder. By default only non-blank fields will be included in the created xml file. Images will be exported to an output folder which will be created adjacent to the output file. Image file names will be formatted on an absolute basis and inserted into field 52.19 of the exported xml. A summary JSON file will be created.

Type 52 Images

Export will insert the name of the each image file into its corresponding type 52 image view data record. The name is stored into field 5.19, which is normally houses the image data. By putting the check image file name into field 52.19, Export is able to logically associate each image with its type 52 record. Note that when using the x9.100-180 standard, images are stored in field 52.27, which is fully supported by this process. .

When the type 52 record is exported, the image field is updated to contain one of the following values, subject to the selected options:

- Absolute file name, which contains the drive and path (folder) of each image file along with the actual file name. It is recommended that you use absolute file names since they fully describe the output file location. Absolute file names are required when the exported file is used by certain internal functions such as Generate.

- Relative file name, which contains the base file name only (it does not include the path). Relative names are useful when the exported CSV and images will be transported to other environments or systems.
- Base64-basic encodes the image using base64 and inserts the resulting string into the image field. This eliminates the external storage of the image in the file system and can simplify access to the image data. Base64 image exports run substantially faster than exports into the file system, since the operating system overhead to update the file system is eliminated. Image size is typically increased by 30-40 percent when using this encoding format.
- Base64-mime is similar to base64-basic encoding, just based on the MIME format.

Exported Image Formats

When exporting images, you can optionally select the image format that will be created, which can be TIF, PNG, JPG, or GIF. It is important to realize that embedded images within x9.37 files are in TIF format. This means that when images are written in TIF format (which is the default), they can be written exactly as they are contained within the x9.37 file and do not need to be converted in any manner. However, export also allows you to select an alternative format. When that is done, images must be converted from the TIF format to the user selected format. Please realize that image conversions can be time consuming, with PNG probably being the better alternative given its performance and compression. PNG and JPG images will retain their original size and DPI. Since GIF images to not have an internally defined DPI, they will be standardized to 200 DPI as a matter of convenience. Image conversions should be utilized carefully and only when absolutely needed. An alternative is to export images in TIF format and then subsequently utilize a batch conversion strategy to convert them later, perhaps when importing them into an archive or user application.

Export Considerations

You can use exported files as input to other applications such as Excel or your proprietary application systems. Export is an excellent tool to allow you to create CSV files that are shareable and can be used in a large variety of ways.

You can optionally browse the created text files using various industry standard tools such as Excel, Libre Office, NotePad or NotePad++. These tools include search facilities allowing you to find data on a string basis.

This table contains all export formats with additional considerations regarding their usage.

	Description	Switch	Importable?	Selected Record Types ?	Comments
(1)	CSV parsed items into fixed fields.	-xf Include the -i switch to write	No	Yes	Items are parsed into individual fields and exported into a fixed column format. This format can be easier to process since the items are populated on a standard basis

	Description	Switch	Importable?	Selected Record Types ?	Comments
		images.			into fixed columns. This item format is constant and does not vary regardless of the file type (forward presentment or returns) and the associated x9 standard (x9.37 versus x9.100-180). Data fields for non-item groups (file headers, cash letter headers, bundle headers, etc) can be optionally included and will be presented in their native format.
(2)	CSV in native x9 format without images.	-xc	Yes, but only if the images are drawn as proxies since the actual images are not exported along with the data, and then also only when all record types are exported.	Yes	This format contains the field values within the selected x9 record types per the associated x9 standard. It provides a full representation of the x9 data content for the selected record types, and a complete data representation of the entire file when all x9 record types are selected. A common usage is to limit the export to certain record types based on need. For example, the export might be limited to the type 01 file header, the type 25 check records, and the type 26-28 addenda records. The output line numbers will match back to the input x9 file when all x9 record types have been selected.
(3)	CSV in native x9 format with images.	-xc with the -i switch to write images.	Yes, but only when all record types have been exported.	Yes	This format contains the field values within the selected x9 record types per the associated x9 standard. The output line numbers will match back to the input x9 file when all x9 record types have been selected. This export file can be modified using various standard editor tools and then

	Description	Switch	Importable?	Selected Record Types ?	Comments
					imported to create a new x9 file which has been changed per user specific requirements. For example, the order of certain record types within the file can be changed; records can be removed; individual fields can be modified. Repair can also be run after the file is imported to correct trailer records when desired.
(4)	CSV record groups into variable columns.	-xg Include the -i switch to write images.	No	Yes	Output is constructed on a record group basis, where each group is the owner record type with all attached subordinate types. This format can be easier to parse since the x9 records for each group are concatenated into a single string which eliminates the need to separate the rows into record groups in your parser. This format has the further benefit that all x9 fields are present. Note that specific record types (of your choosing) can be excluded and will not appear within the output. For example, you can exclude record types 50-52 if that data is not needed.
(5)	XML.	-xml	No	Yes	Our standard XML format will be written. There are numerous third party XML translators that can be used to convert this output to meet your schema requirements.
(6)	Tiff tag information.		No	No	This export format provides a vision into the tiff tags that are present for each image and is useful when performing a detailed image analysis for a specific originator or capture

	Description	Switch	Importable?	Selected Record Types ?	Comments
					system.
(7)	X9 data.		No	Yes	X9 records are exported in their text (txt) form, which provides access to their record data (eg, typically 80 bytes long) in the exact format as present on the x9 file. When using this export format, there is an option to append the record number as either a suffix or prefix to the written data. Appending the record number allows you to sort the data on various fields and still be able to resort the records into their original order. The record number also allows you to trace every record back to the original location within the original x9 file.
(8)	XML.		No	Yes	Export to XML builds an output xml file that can be processed by other xml enabled applications. Please advise if you have requirements for this functionality and we would be glad to work with you on building new xml formats for specific vendor applications.
(9)	Errors.		N/A	N/A	Errors for the current x9 file are exported and shared with others. All selection criteria apply (x9 record number range and record types). The export can be limited by type (data or image) or severity (error, warn, and info). Output is in a fixed format which can be easily analyzed.

Type 52 Records

Export will insert the name of each image file into its corresponding type 52 image view data record. The name is stored into field 5.19, which is normally houses the image data. By putting the check image file name into field 52.19, Export is able to logically associate each image with its type 52 record.

When the type 52 record is updated, the check image file name can be stored on either an "absolute" or "relative" basis. Absolute file names will contain the drive and path (folder) of each image file along with the actual file name, while relative file names do not include their path. It is recommended that you use absolute file names since they fully describe the output file location. Absolute file names are required when the exported file is used by certain internal functions such as Generate. However, you can also elect to use relative file names, which would allow you to transport the CSV and images to other folder structures and still be able to import the results.

Export as Items into Fixed Columns

Items can be parsed into logical field content and then exported into fixed columns. This format can be easier to parse since the location of individual data columns will be fixed which can simplify your subsequent parsing of this data. Specifically, the type 25 and type 31 records are parsed into logical items that will contain the following columns:

Columns	Data Content
1	Record type
2	Amount
3	Item sequence number
4	MICR Routing
5	MICR OnUs
6	MICR Auxiliary OnUs
7	MICR EPC
8	Documentation type indicator
8	Returns Acceptance Indicator
10	MICR valid indicator
11	BOFD indicator
12	Addendum count
13	Correction indicator
14	Archive type indicator
15	Credit account
16	Return reason

Columns	Data Content
17	Forward bundle date
18	Return notification indicator
19	Payor bank name
20	Payor bank business date
21	Payor account name
22	Field4 parsed from the item MICR OnUs field
23	Account parsed from the item MICR OnUs field
24	Process Control parsed from the item MICR OnUs field
25	Check number, which is assigned first from MICR AuxOnUs (when populated), from MICR Process Control (when that field contains four or more numeric digits), and is otherwise not populated
26	Reserved-1
27	Reserved-2
28	Image creator date from the front image
29	Image creator routing from the front image
30	Image reference key from the front image
31	Front image name or multi-page image name (when exported).
32	Back image name (when exported)
33-45	First primary (26 or 32) or secondary (28 or 35) endorsement
33	Record type
34	Routing
35	Endorsement date
36	Item sequence number
37	Deposit account number
38	Deposit branch
39	Payee name
40	Truncation indicator
41	Conversion indicator
42	Return reason
43	Endorsing bank identifier
44	User field
46-58	Second endorsement

Columns	Data Content
59-71	Third endorsement
72-84	Fourth endorsement
Etc	Continued for as many endorsements that exist for this item

Notes on Check Number

Check number will be in one of two fields depending on the data that is present, which is largely based on check size. There are two physical check sizes (business checks and wallet checks). Typically, business checks are 8+ inches wide while wallet (consumer) checks are 6 inches wide. Because of the larger width associated with business checks, they can encode the MICR AuxOnUs field which is where they can place a check number that is up to 15 digits long. This larger field is needed to accommodate the long check number that is required by business accounts. In our fixed field export, the AuxOnUs field is present in CSV column 6.

Retail (consumer) checks utilize the wallet format and do not have encoding space for the AuxOnUs field. Because of this, their check number is more typically placed in the MICR OnUs field next to the account number. For example, the MICR OnUs field can be found as formatted “accountNumber/processControl”, where the process control field contains the check number and is typically four digits. If the process control field contains less then four digits, then it is more likely a transaction code and not a check number. The bottom line is that although this is a common practice utilized by many financial institutions, it is not an absolute standard so you there will be variations from this implementation. In our fixed field export, the MICR OnUs field is present in CSV column 5 and the parsed Process Control field is in fixed field 24.

Check number would then typically be found in one of these two fields. The business logic to extract the check number would be as follows:

- If MICR AuxOnUs is populated (not blank), then it contains the check number.
- Otherwise, if Process Control is populated (not blanks) and contains four or more digits, then that field contains the check number.

Notes on Check Date

The data that a check was physically written is not present in the x9.37 data and is only present in hand writing within the front side check image.

The best data that is present within the x9.37 standards for check data would instead be the date that the item was presented to the bank of first deposit. If the check is deposited on a timely basis, then this presentment date from the BOFD addenda record is an approximation of the check date, subject to the data that is available within the x9.37 data fields. In our fixed field export, the BOFD addenda date from the type 26 record is in CSV column 35.

Export as Native X9 Format

Native format follows the current x9 specification. For example, an input x9 file encoded per the x9.37 DSTU specification will be exported per the record layouts that are defined by that standard. You must reference the associated x9 specification to obtain a list of the fields that are exported by record type. These fields will be the same as displayed within the X9Assist viewers. This export format has the advantage that it covers the entire extent of the x9 file specification.

When using this export format, there is an additional option to append the record number as either a suffix or prefix to the written data. Appending the record number allows you to sort the data on various fields and still be able to resort the records into their original order. The record number also allows you to trace every record back to the original location within the original x9 file.

Export as Groups into Variable Columns

Records can be exported as groups and not individual record types. A record group consists of the owner record type (for example, a type 25 or type 31) followed by all records that are attached to that owner. Although the concept of record groups only applies to items, it also logically applies to other record types when they are the owner of type 68 user records. For example, a record group might consist of only a single type 01 file header, but it could also consist of a type 01 file header followed by two type 68 user records.

Record groups are exported as a CSV of all record types and fields within the group. For an item, this single CSV row might contain fields for the following record types: 25, 26, 28, 52, 54, 52, and 54. Also remember that record types can be excluded from this process. For example, you can exclude record types 50 and 52 from the export, and in that case this same record group would be exported as: 25, 26, 28, and 28. If you only need the item with the attached BOFD endorsement, then you can exclude 28, 50 and 52, and then the exported record group will consist of just the 25 and 26 records. Although these are examples of forward presentment files, the process applies equally to return files.

This export format is provided as a convenience since it may be easier to parse by your application programs. Although the concept of record groups is shared with item export, it has the benefit that it includes every field with the individual record types will be exported.

Export the Type99 Trailer Record

There are situations where there is a need to read an input x9 file and export only the type 99 file control trailer record to CSV. This can be achieved using the -xc and -rectypes:99 command line parameters. The result will be an output CSV file that contains a parsed representation of the type 99 record, which contains file totals. Similarly, -rectypes:”01|99” will limit the export to the file header and file control trailer records.

In addition to this to this approach, you can add either the -t (output text file) or -x (output xml file) command line parameters to various x9utilities export functions, which will create a summary file with file totals.

Export as Errors

Errors can be exported in CSV format to allow you to get a list of errors associated within a given X9 file that you can easily share with others. All of the previously stated selection criteria applies (x9 record number range and record types).

In addition, error export allows you to indicate that you want to limit the export based on:

- All errors
- X9 errors only
- Image (tiff) errors only

You can also identify the severity of the errors to be exported. This can be all error severities, or combinations of error level, warn level, and info level. The field data that is written is aligned per the columns as depicted on the Errors tab. This list of fields is as follows:

1) Error description
2) Error identifier
3) Error record number
4) Error field number
5) Error field name
6) Error field x9 data position
7) Error field x9 data length
8) Error field value
9) Error field mandatory indicator
10) Error field list of allowable values
11) Error field primary edit rule
12) Supplemental information
13) Item amount
14) Item ECE sequence number
15) Item routing number
16) Cash letter record number
17) Cash letter identifier
18) Cash letter amount
19) Cash letter business date
20) Bundle record number
21) Bundle amount
22) BOFD routing number
23) BOFD business date
24) BOFD sequence number

Sample CSV output (which is the default format)

```
01,03,"T",123456780,123456780,20140810,1201,"N","VIEW","VIEW","A",,,
10,01,123456780,123456780,20140808,20140810,1201,"I","G",1,"X9Assist",,"C",,
20,01,123456780,123456780,20140808,20140810,57000000,1,,123456780,,
25,,,08777070,6,"29602722/5526",0000010002,44000001,"G",8,1,"Y",01,0,"B"
```

26,1,123456780,20140807,44000001,,,,"Y",0,,,
50,1,087770706,20140807,00,00,0006302,0,00,0,,,,,0,,
52,123456780,20140808,,44000001,,,,0,,,,,0000,,0,,0006302,"c:/users/x9ware5/documents/
x9ware/fileUtilities/Test file with 2 checks
exporter_IMAGES/Bundle_000003/Image_000007_amount_10002_front.tif"
50,1,087770706,20140807,00,00,0001865,1,00,0,,,,,0,,
52,123456780,20140808,,44000001,,,,0,,,,,0000,,0,,0001865,"c:/users/x9ware5/documents/
x9ware/fileUtilities/Test file with 2 checks
exporter_IMAGES/Bundle_000003/Image_000009_amount_10002_back.tif"
25,,,09777059,2,"60333044/5587",0000010004,44000002,"G",8,1,"Y",01,0,"B"
26,1,123456780,20140807,44000002,,,,"Y",0,,,
50,1,097770592,20140807,00,00,0006679,0,00,0,,,,,0,,
52,123456780,20140808,,44000002,,,,0,,,,,0000,,0,,0006679,"c:/users/x9ware5/documents/
x9ware/fileUtilities/Test file with 2 checks
exporter_IMAGES/Bundle_000003/Image_000013_amount_10004_front.tif"
50,1,097770592,20140807,00,00,0001865,1,00,0,,,,,0,,
52,123456780,20140808,,44000002,,,,0,,,,,0000,,0,,0001865,"c:/users/x9ware5/documents/
x9ware/fileUtilities/Test file with 2 checks
exporter_IMAGES/Bundle_000003/Image_000015_amount_10004_back.tif"
70,0002,000000020006,000000020006,00004,,
90,000001,00000002,00000000020006,000000004,"File Generator",20140810,
99,000001,00000018,00000002,0000000000020006,,,

XML Flat Format Example (created using the -xmlf switch)

This is sample type 25 record as exported into flat format:

```
<record><type>25</type><index>25</index><name>Check Detail Record</name>
  <field><number>25.1</number><name>Record Type</name><value>25</value></field>
  <field><number>25.4</number><name>Payor Bank Routing
      Number</name><value>07777039</value></field>
  <field><number>25.5</number><name>Payor Bank Routing Number Check
      Digit</name><value>2</value></field>
  <field><number>25.6</number><name>MICR
      On-Us</name><value>29343913/114</value></field>
  <field><number>25.7</number><name>Amount</name><value>0000010006</value></field>
  <field><number>25.8</number><name>ECE Institution Item Sequence
      Number</name><value>44000003</value></field>
  <field><number>25.9</number><name>Documentation Type
      Indicator</name><value>G</value></field>
  <field><number>25.10</number><name>Return Acceptance
      Indicator</name><value>3</value></field>
  <field><number>25.11</number><name>MICR Valid
      Indicator</name><value>1</value></field>
  <field><number>25.12</number><name>BOFD Indicator</name><value>Y</value></field>
  <field><number>25.13</number><name>Check Detail Record Addendum
      Count</name><value>02</value></field>
  <field><number>25.14</number><name>Correction
      Indicator</name><value>3</value></field>
```

```
   <field><number>25.15</number><name>Archive Type
         Indicator</name><value>D</value></field>
 </record>
```

XML Hierarchical Format Example (created using the -xmlh switch)

This is sample type 25 record as exported into hierarchical format:

```
<type-25><index>4</index><name>Check Detail Record</name>
      <RecordType><number>25.1</number><value>25</value></RecordType>
      <PayorBankRoutingNumber><number>25.4</number><value>05777093</value></
         PayorBankRoutingNumber>
      <PayorBankRoutingNumberCheckDigit><number>25.5</number><value>0</value></
         PayorBankRoutingNumberCheckDigit>
      <MICROn-Us><number>25.6</number><value>209 153 53/111</value></MICROn-Us>
      <Amount><number>25.7</number><value>0000010000</value></Amount>
      <ECEInstitutionItemSequenceNumber><number>25.8</number><value>44000000</value></
         ECEInstitutionItemSequenceNumber>
      <DocumentationTypeIndicator><number>25.9</number><value>G</value></
         DocumentationTypeIndicator>
      <ReturnAcceptanceIndicator><number>25.10</number><value>0</value></
         ReturnAcceptanceIndicator>
      <MICRValidIndicator><number>25.11</number><value>1</value></MICRValidIndicator>
      <BOFDIndicator><number>25.12</number><value>Y</value></BOFDIndicator>
      <CheckDetailRecordAddendumCount><number>25.13</number><value>02</value></
         CheckDetailRecordAddendumCount>
      <CorrectionIndicator><number>25.14</number><value>0</value></CorrectionIndicator>
      <ArchiveTypeIndicator><number>25.15</number><value>A</value></ArchiveTypeIndicator>
</type-25>
```

ExportCsv

ExportCsv provides a variation on the capabilities that are offered by the alternative Export function. ExportCsv is especially useful when you need to define specific csv output columns, or embed the export parameters within an xml control file.

ExportCsv field names

Fields to be exported are defined using their "record dot field" name. For example, 25.7 is the type 25 amount field. Fields can be selected from the following record types:

- Type 01 – file header
- Type 10 – cash letter header
- Type 20 – bundle header
- Type 25/31/61/62 – debit or credit records
- Type 50 – front image view detail record
- Type 52 – front image view data record

Special field names

In addition to the "record dot field" name format, there are several special field names that can be used for specific conditions:

Field Name	Content
MicrRouting	Combination of the item 8 digit routing plus the 1 digit check it.
AccountNumber	Account number as extracted from the MICR OnUs field.
CheckNumber	Check number, which is first taken from AuxOnUs (when populated) and otherwise extracted from MICR OnUs. Check numbers within MICR OnUs must be at least four (4) digits in length. If your application requires MICR OnUs check numbers that are shorter than 4 digits, then you should instead write your own extraction logic using the provided fields.
ReturnReason	Best return reason extracted from the item addenda records.
FrontImage	Front image.
BackImage	Back image.

Image formats

Images are typically always exported into TIFF format, since that is the image exchange format. However, several other formats are available. These should be used carefully, since conversions can be imperfect, and they will always take additional time. The image formats that are available are as follows:

- tif
- png
- jpg
- gif

Image methods

In addition to the “record dot field” name format, there are several special field names that can be used for specific conditions:

Image Method	Functionality
a	Absolute (fully qualified) file name.
r	Relative file name (contains only the base part of the fully qualified name).
b	Base64 basic format (the image will be embedded within the csv).
m	Base64 mime format (the image will be embedded within the csv).

Command line options

Switch	Description
-j	A summary JSON file will be created with a suffix of “_summary.json” in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of “_summary.txt” in the same folder as the output CSV or XML file.
-x	A summary XML file will be created with a suffix of “_summary.xml” in the same folder as the output CSV or XML file.
-l	Will list all x9 records to the system log.
-xctl:	Specifies the name of the export xml control file to be loaded, which define all available export formats. In the most simple of cases, the control xml control file will contain a single output format. However, in more complex cases, you might have a separate output format for various specific purposes (perhaps by downstream application system, or perhaps for each customer that is to receive these files). Each output format within the xml control file is given a name, which should correspond to the purpose.
-xfmt:	Specifies the name of the export format definition to be used within the export control file. This is a string value that might be defined as “customer3” or “imageArchive”. This name must represent an output format entry with the export xml control file. The format defines the columns to be created, the name of the output csv file to be written, the name of the output image folder which will be created, and the various xml parameters that are used to control the export process.

Command line examples

x9util -exportCsv -xctl:<xmlExportControlFile>” -xfmt:<xmlExportFormat> <input.x937>

reads <input.x937> and creates an output csv with the user defined columns. The output csv file name is defined within the xml export control file.

x9util -exportCsv -xctl:<xmlExportControlFile>" -xfmt:<xmlExportFormat> <input.x937> <output.csv> <imageFolder>

reads <input.x937> and creates <output.csv> with the user defined columns. The output file and image folder names are taken from the xml export control file first and the command line second.

x9util -t -exportCsv -xctl:<xmlExportControlFile> -xfmt:<xmlExportFormat> <input.x937>

reads <input.x937> and creates file and output csv with the user defined columns. Summary TXT file will be created

ExportCsv XML Definition

The ExportCsv XML file is used to customize the columns that are created when a file is exported to CSV. It allows you to including only those columns that are needed by your application, and also to define the order that those fields will appear in the output CSV file. Each of these XML definitions are completely self-contained, with the output file name and all associated parameters as required for the CSV export operation. This approach minimizes command line parameters and instead locates those within the XML definition. This XML file can include the following tags:

XML Tag	Content	Comments
<exportName>	Name of this export format.	Must be unique within this overall definition.
<csvFileName>	Fully qualified output csv file name to be created.	
<imageFolder>	Fully qualified folder where images will be written. A sub-folder will be created within this folder using the input file name. Additional folders will then be created at the bundle level.	
<configName>	"auto" or a valid configuration name. Examples are x9.37, x9.100-187-2008, etc.	auto=automatically assigned based on file header inspections
<dateTimeStamp>	A date pattern to be utilized when output segments are to be suffixed with a time stamp. This facility can be used to make all output file names unique. A commonly used assignment would be: yyyyMMdd_HHmmss . You can do an internet search on "Java Date Format Pattern" for all allowable pattern characters and usage examples.	Omitted.

XML Tag	Content	Comments
<doNotRewrite>	Indicates if an existing output segment can be overwritten. Values can be true or false.	False.
<clearImageFolder>	Determines if the output image folder should be cleared. Values are true or false.	true
<includeColumnHeaders>	Determines if column header names should be included as row one. Values are true or false.	false
<includeDecimalPoints>	Determines if decimal points should be inserted into amounts. Values are true or false.	false
<summaryTxtFile>	Determines if the summary txt file (which contains totals from the input file) should be created. Values are true or false.	false
<summaryXmlFile>	Determines if the summary xml file (which contains totals from the input file) should be created. Values are true or false.	false
<imageFormat>	Defines the output image format. The TIFF format is highly recommended, since it allows the images to be copied exactly as they exist on the input file. Note that image conversions will add substantially to run time.	tif; alternative values are png, jpg, and gif; an empty string (eg, <imageFormat/> turns image export off.
<imageMethod>	Defines how images are exported. This can be either as a file name that is included in the column (“a” or “r”) or as a base64 string that is inserted into the column (“b” or “m”).	a=absolute file name, r=relative file name, b=base64 basic, m=base64mime
<fields>	A list of all output CSV fields that will be created.	
<field>	<fields> child tag that defines the record dot field for the next column to be exported.	
<columnName>	<fields> child tag that defines the column name to be assigned for this field (in row one) when <includeColumnHeaders> is true.	Defaults to field name per the rules specification.

Sample XML for Forward Presentment and Returns

```
<?xml version="1.0" encoding="UTF-8"?>
<exportCsv>
  <formats>
    <format>
            <exportName>exportForwardPresentment</exportName>
            <csvFileName>C:\Users\X9Ware5\Documents\x9_assist\files_Utilities\exportTest.csv</csvFileName>
            <imageFolder>C:\Users\X9Ware5\Documents\x9_assist\files_Utilities\exportedImages</imageFolder>
            <configName>auto</configName>
            <dateTimeStamp></dateTimeStamp>
            <doNotRewrite>false</doNotRewrite>
            <clearImageFolder>true</clearImageFolder>
```

```
        <includeColumnHeaders>true</includeColumnHeaders>
        <includeDecimalPoints>true</includeDecimalPoints>
        <summaryTxtFile>true</summaryTxtFile>
        <summaryXmlFile>true</summaryXmlFile>
        <imageFormat>tif</imageFormat>
        <imageMethod>a</imageMethod>
        <fields>
          <output> <field>1.4</field> </output>
                <output> <field>1.5</field> </output>
                <output> <field>10.3</field> </output>
          <output> <field>10.4</field> </output>
                <output> <field>20.5</field> </output>
                <output> <field>20.10</field> </output>
                <output> <field>25.8</field> </output>
                <output> <field>25.7</field> <columnName>ItemAmount</columnName></output>
                <output> <field>MicrRouting</field> </output>
                <output> <field>25.3</field> </output>
                <output> <field>25.2</field> </output>
                <output> <field>25.6</field> </output>
                <output> <field>26.3</field> </output>
                <output> <field>26.4</field> </output>
                <output> <field>26.8</field> </output>
                <output> <field>50.3</field> </output>
                <output> <field>ReturnReason</field> </output>
                <output> <field>FrontImage</field> </output>
                <output> <field>BackImage</field> </output>
        </fields>
</format>

<format>
        <exportName>exportReturns</exportName>
        <csvFileName>C:\Users\X9Ware5\Documents\x9_assist\files_Utilities\exportTest.csv</csvFileName>
        <imageFolder>C:\Users\X9Ware5\Documents\x9_assist\files_Utilities\exportedImages</imageFolder>
        <configName>auto</configName>
        <dateTimeStamp></dateTimeStamp>
        <doNotRewrite>false</doNotRewrite>
        <clearImageFolder>true</clearImageFolder>
        <includeColumnHeaders>true</includeColumnHeaders>
        <includeDecimalPoints>true</includeDecimalPoints>
        <summaryTxtFile>true</summaryTxtFile>
        <summaryXmlFile>true</summaryXmlFile>
        <imageFormat>tif</imageFormat>
        <imageMethod>a</imageMethod>
        <fields>
          <output> <field>1.4</field> </output>
                <output> <field>1.5</field> </output>
                <output> <field>10.3</field> </output>
          <output> <field>10.4</field> </output>
                <output> <field>20.5</field> </output>
                <output> <field>20.10</field> </output>
                <output> <field>31.10</field> </output>
                <output> <field>31.5</field> <columnName>ItemAmount</columnName></output>
                <output> <field>MicrRouting</field> </output>
                <output> <field>31.11</field> </output>
                <output> <field>33.3</field> </output>
                <output> <field>31.4</field> </output>
                <output> <field>32.3</field> </output>
                <output> <field>32.4</field> </output>
                <output> <field>32.8</field> </output>
                <output> <field>50.3</field> </output>
                <output> <field>ReturnReason</field> </output>
                <output> <field>FrontImage</field> </output>
```

```
                    <output> <field>BackImage</field> </output>
            </fields>
    </format>
  </formats>
</exportCsv>
```

Sample ExportCsv Output File

ImmediateDestinationRoutingNumber,ImmediateOriginRoutingNumber,DestinationRoutingNumber,ECEInstitutionRoutingNumber
,BundleBusinessDate,ReturnLocationRoutingNumber,ECEInstitutionItemSequenceNumber,ItemAmount,MicrRouting,ExternalProc
essingCode,AuxiliaryOnus,MICROn-
Us,Amount,ECEInstitutionItemSequenceNumber,BankofFirstDepositRoutingNumber,BOFDBusiness(Endorsement)Date,PayeeNa
me,ImageCreatorRoutingNumber,ReturnReason,FrontImage,BackImage
123456780,123456780,123456780,123456780,20180809,123456780,44000000,00000100.00,057770930,,,20915353/111,00000100
.00,44000000,123456780,20180807,Cash,123456780,,"C:/Users/X9Ware5/Documents/x9_assist/images/Test
ICL_images/Bundle_000003/Image_000008_amount_10000_front.tif","C:/Users/X9Ware5/Documents/x9_assist/images/Test
ICL_images/Bundle_000003/Image_000010_amount_10000_back.tif"
123456780,123456780,123456780,123456780,20180809,123456780,44000001,00000100.02,087770706,,,29602722/112,00000100
.02,44000001,123456780,20180807,Cash,123456780,,"C:/Users/X9Ware5/Documents/x9_assist/images/Test
ICL_images/Bundle_000003/Image_000015_amount_10002_front.tif","C:/Users/X9Ware5/Documents/x9_assist/images/Test
ICL_images/Bundle_000003/Image_000017_amount_10002_back.tif"

Import

Import reads a CSV file and then writes the individual data records to an output x9 file.

Import is the exact opposite of our Export tool You can use Export output as Import input.

Field 52.19 must contain the fully qualified image name, enclosed in quote marks, with forward file separators. Please refer to the previous Export example for proper format of the type 52 records.

Command line options

Switch	Description
-config:	Use a specific x9 configuration which defaults to “x9.37”. For example, the CPA 015 standard can be selected as: “-config:x9.CPA_015”.
-r	X9 totals will be automatically recalculated and replaced in the output x9 file. All incoming totals in those trailer records will be ignored.
-j	A summary JSON file will be created with a suffix of “_summary.json” in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of “_summary.txt” in the same folder as the output x9 file.
-x	A summary XML file will be created with a suffix of “_summary.xml” in the same folder as the output x9 file.
-l	Will list all x9 records to the system log.
-dpi:	Assigns the output image dpi used to draw images, with a default of 240. Assigned values must be either 200 or 240.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -import <input.csv>

reads <input.csv> and creates file <input.x937> within the designated input folder.

x9util -import <input.csv> <output.x937>

reads <input.csv> and creates file <output.x937> within the designated output folder.

x9util -import <input.csv> <output.x937> <imageFolder>

reads <input.csv> and creates file <output.x937> within the designated output folder using images from <imageFolder> as needed and only when relative image file names are present.

x9util -import -x <input.csv> <output.x937>

reads <input.csv> and creates file <output.x937> within the designated output folder. A summary XML file will be created.

x9util -import -j <input.csv> <output.x937>

reads <input.csv> and creates file <output.x937> within the designated output folder. A summary JSON file will be created.

Excessive Field Sizes

Import will set an exit status of 3 if there are any fields that have been assigned values that exceed their maximum defined size, per the x9.37 standards. An example would be an attempt to assign MICR OnUs with a 22 character value, or MICR AuxOnUs a 17 character value. In these situations, the input values will be truncated and the error condition included in the system log, along with the assigned exit status.

Validate

Validate reads an x9 file in industry defined formats and applies x9 data and image validations against the parsed records. Identified validation errors are written to an output CSV file.

The x9 configuration which is used for the validation process is provided on the command line. Since field level validations can vary widely across the various x9 specifications, the errors that are identified can vary substantially subject to which x9 specification is applied to the file. Generally, the x9.37 specification is the most lenient of standards, which means that it will generate the fewest number of errors against any given file. However, when running a validation, you need to consider and apply the specific x9 configuration which best meets your application requirements.

See topic “Supported x9 Configurations” for configurations that are supported by validate.

Command line switches (parameters) are used to further control the Validate operation. By default, the x9.37 rules will be applied to the x9 file during validation. However, you can also use a command line switch to apply an alternate x9 configuration for the validation process.

Command line options

Switch	Description
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific “-config:” value. For example, the CPA 015 standard can be selected as: “-config:x9.CPA_015”.
-j	A summary JSON file will be created with a suffix of “_summary.json” in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of “_summary.txt” in the same folder as the input x9 file.
-x	A summary XML file will be created with a suffix of “_summary.xml” in the same folder as the input x9 file.
-l	Will list all x9 records to the system log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -validate <input.x937>

validates <input.x937> and creates error file <input.csv> within the designated input folder.

x9util -validate <input.x937> <output.csv>

validates <input.x937> and creates error file <output.csv> within the designated output folder.

x9util -x -validate <input.x937> <output.csv>

validates <input.x937> and creates error file <output.csv> within the designated output folder. A summary XML file will be created.

x9util -j -validate <input.x937> <output.csv>

validates <input.x937> and creates error file <output.csv> within the designated output folder. A summary JSON file will be created.

Exit status

Validate will set the overall run exit status as follows:

- -3 = input file not found
- -2 = invalid function
- -1 = aborted
- 0 = run successfully with no errors
- 1 = run successfully with informational message(s)
- 2 = run successfully with warning message(s)
- 3 = run successfully with error message(s)
- 4 = run successfully with severe error message(s)

Error File Columns

The output csv error file contains the following columns:

Column	Content	Notes
1	X9 record number	Record number within the x9 file where the error occurred.
2	Record type	X9 record type or 8888 if the error is related to an image within a type 52 record.
3	Field number	X9 field number or 9999 if the error.
4	Field name	Field name associated with the error.
5	Error name	Internally assign error name. A complete list of all errors can be found in /xml/messages/messages.xml. Be advised that that error messages and potentially their names can change from release to release of the SDK.
6	Error severity	Error severity (severe, error, warn, or info).
7	Message text	Error message text.
8	Supplemental Information	Supplemental information concerning this specific error.

Error File example

1,1,14,"Reserved","reservedMustBeBlank",2,"Reserved field must be blank per current x9 rules; invalid character(1)","invalid character(1)"
10,8888,278,"RowsPerStrip","tiffMultiStripImagesNotAllowed",3,"Tiff multi-strip images are not allowed; number of strips(83.375) tag(278) typeString(SHORT_LONG) type(13) count(1) rule(tag278RowsPerStrip)","number of strips(83.375) tag(278) typeString(SHORT_LONG) type(13) count(1) rule(tag278RowsPerStrip)"
15,8888,282,"xResolution","tiffDPIisIncorrectPerRules",1,"Tiff image DPI (dots per inch) is incorrect per xml rules; xDpiValue(220) tag(282) typeString(RATIONAL) type(5) count(1) value(=200|240)","xDpiValue(220) tag(282) typeString(RATIONAL) type(5) count(1) value(=200|240)"
15,8888,283,"yResolution","tiffDPIisIncorrectPerRules",1,"Tiff image DPI (dots per inch) is incorrect per xml rules; yDpiValue(220) tag(283) typeString(RATIONAL) type(5) count(1) value(=200|240)","yDpiValue(220) tag(283) typeString(RATIONAL) type(5) count(1) value(=200|240)"
17,8888,282,"xResolution","tiffDPIisIncorrectPerRules",1,"Tiff image DPI (dots per inch) is incorrect per xml rules; xDpiValue(220) tag(282) typeString(RATIONAL) type(5) count(1) value(=200|240)","xDpiValue(220) tag(282) typeString(RATIONAL) type(5) count(1) value(=200|240)"
17,8888,283,"yResolution","tiffDPIisIncorrectPerRules",1,"Tiff image DPI (dots per inch) is incorrect per xml rules; yDpiValue(220) tag(283) typeString(RATIONAL) type(5) count(1) value(=200|240)","yDpiValue(220) tag(283) typeString(RATIONAL) type(5) count(1) value(=200|240)"
22,8888,9999,"End of Facsimile Block","tiffImageSegmentDoesNotEndEofb",3,"Tiff image segment does not end with EOFB; end of image segment(888888FFFFFFFFFFFC0040000)","end of image segment(888888FFFFFFFFFFFC0040000)"
24,8888,9999,"End of Facsimile Block","tiffImageSegmentDoesNotEndEofb",3,"Tiff image segment does not end with EOFB; end of image segment(1FFFFFFFFFFFFFFFFFFE002000)","end of image segment(1FFFFFFFFFFFFFFFFFFE002000)"
29,8888,256,"ImageWidth","tiffWidthGreaterThanMaximum",1,"Tiff image width greater than maximum inches; width(11.0) greater than maximum(10.5)","width(11.0) greater than maximum(10.5)"
31,8888,256,"ImageWidth","tiffWidthGreaterThanMaximum",1,"Tiff image width greater than maximum inches; width(11.0) greater than maximum(10.5)","width(11.0) greater than maximum(10.5)"
4,25,13,"Check Detail Record Addendum Count","endorsementAddendumCountIncorrect",1,"Endorsement count does not agree with addenda present; value(3) expecting(2)","value(3) expecting(2)"
694,8888,0,"Image Header","tiffMoreThanOneImage",1,"Tiff contains more than one image",
701,8888,0,"Image Header","tiffImageNot49Or4d",1,"Tiff image invalid; does not begin 0x49492a00 or 0x4d4d002a; invalid endian identifier(0x0000)","invalid endian identifier(0x0000)"
703,8888,0,"Image Header","tiffImageNot49Or4d",1,"Tiff image invalid; does not begin 0x49492a00 or 0x4d4d002a; invalid endian identifier(0x0000)","invalid endian identifier(0x0000)"
706,99,7,"Immediate Origin Contact Phone Number","notNumeric",1,"Not numeric; editRuleFailed(n)","editRuleFailed(n)"
8,8888,278,"RowsPerStrip","tiffMultiStripImagesNotAllowed",3,"Tiff multi-strip images are not allowed; number of strips(83.375) tag(278) typeString(SHORT_LONG) type(13) count(1) rule(tag278RowsPerStrip)","number of strips(83.375) tag(278) typeString(SHORT_LONG) type(13) count(1) rule(tag278RowsPerStrip)"

Custom Rules

Validation of x9.37 files is complicated by several factors. There are several standards, so the validation must invoke the proper standard for the input file. This is not an issue when a single standard is being used for all processing. However, if multiple standards can be accepted, then selection of the standard can be a more complicated issue. This assignment can be made subject to the source of the file. Another alternative is allows validation to automatically choose the validation rules, which can be selected using -config:auto.

Validation is further complicated by the fact that standards are loosely implemented by many originators, which results in processors accepting many files that have minor validation errors. This makes the process of accepting files based on validation results very difficult, since the reality is that most organizations will need to accept files that contain certain types of errors.

Once a decision is made to validate files, you will then need an approach to determine which validation errors are immaterial versus critical. This ultimately comes down to a filtering process that must be applied against the errors that are thrown by validation. There are several ways this can be accomplished:

- Develop your own list of the errors to be filtered (ignored) using the error name. You can add to this list based the experience gained from your own testing. Error names are displayed by X9Assist, so you can use our desktop software to validate files within your environment, as you build this list. This is the most controllable design.
- Use the X9Assist Message Editor to build a custom message file. The Message Editor constructs an override file which is applied to the standard message set. The severity of messages can be reduced from error to warn, info, or even ignore. By reducing the severity associated with specific errors, you can ultimately use the exit status to make your decision as to whether a file passes your minimum validation requirements. You will also need to use the X9Assist Configuration Editor to build your own configuration that includes your custom message set.
- Use the X9Assist Rules Editor to build a custom set of validation rules. The rules editor constructs either a rules definition that can either be a basis or extension. A basis is a complete set of rules, while an extension is an override to an existing basis rule set. When creating custom rules, you will also need to use the X9Assist Configuration Editor to build your own configuration that references your custom rules definition. Be advised that the process of building custom rules is an advanced topic.

Make

Make reads an input use case file and applies our standard Make/Generate process (as implemented in X9Assist) to manufacture an output x9 file. X9Utilities -make can process against either CSV or Excel use case files, just like our X9Assist desktop product.

The x9 configuration which is used by Make can be provided either in the generator xml or alternatively on the command line. The initial assignment is made using the generator, since that follows the same procedure implemented within X9Assist.

In a similar fashion, the routing list file that is used by Make can be provided either in the generator xml or alternatively on the command line. The initial assignment is made using the generator, since that follows the same procedure implemented within X9Assist.

See topic “Supported x9 Configurations” for configurations that are supported by Make.

Make requires actual dates which are used to populate various fields in the output x9 file. These date related parameters can be defined explicitly within the generator, or can be refreshed based on the current date. This is the same funtionality as provided by X9Assist Make. When date values are provided, they must be in YYYYMMDD format. The associated generator fields are:

- Boolean dateRefresh = true;
- Date businessDate;
- Date createDate;
- Integer createTime = X9GenerateXml937.DEFAULT_TIME;
- Date itemStartDate;
- Date itemEndDate;
- Date primaryAddendumDate;
- Date secondaryAddendumDate;

Command line switches (parameters) are used to further control the Validate operation. By default, the x9.37 rules will be applied to the x9 file during validation. However, you can also use a command line switch to apply an alternate x9 configuration for the validation process.

Command line options

Switch	Description
-reformatter	The fully qualified file name of the reformatter xml definition to be loaded.
-generator	The fully qualified file name of the generator xml definition to be loaded.
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific “-config:” value. For example, the CPA 015 standard can be selected as: “-config:x9.CPA_015”.
-j	A summary JSON file will be created with a suffix of “_summary.json” in the same

Switch	Description
	folder as the output x9 file.
-t	A summary text file will be created with a suffix of "_summary.txt" in the same folder as the input x9 file.
-x	A summary XML file will be created with a suffix of "_summary.xml" in the same folder as the input x9 file.
-l	Will list all x9 records to the system log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -make -j -reformatter:"reformatter.xml" -generator:"generator.xml"
<useCases.csv> <output.x9>

runs make/generate with your reformatter, generator, and use case input file, to create the designated output file. The routing list to be used by Make must be defined in the generator.

x9util -make -j -reformatter:"reformatter.xml" -generator:"generator.xml"
<useCases.csv> <routingList.csv> <output.x9>

runs make/generate with your reformatter, generator, use case input file, and routing list csv file, to create the designated output file.

Exit status

Make will set the overall run exit status as follows:

- -3 = input file not found
- -2 = invalid function
- -1 = aborted
- 0 = run successfully with no errors

Merge

Merge combines one or more x9 input files into x9 output file(s). Merge is a robust process which can be run against a potentially large number of input x9 files. Eligible input files are selected from an input "landing zone" which is identified on the command line. Merge is designed to be run periodically against the landing zone, most probably as a scheduled task, where all current content of the input folder will be merged into one or more output x9 files. In support of that design, merge operates in the following manner:

- The landing zone can contain files, folders, or a combination of files and folders. Merge selects files from the input folder based on extension to identify candidate files that are to be merged. Merge can also optionally drill down into subfolders (using the "sf" command line switch). This implementation is essentially can be based on folder of folders, which allows the landing zone to be organized as needed to control the incoming files based on your requirements and to keep the number of files within folders to a reasonable level.
- The input folder command line parameter identifies the location of the landing zone. The folder name should be specified as a quoted string using forward slash separators. Backslashes are not recommended since they are part of an ESCAPE character sequence and can be unexpectedly ignored. The folder name should not end with the separator character. We highly recommend enclosing the input folder within quote " marks, which are mandatory when the folder name contain embedded blanks.
- Merge has a command line option to operate at either the cash letter or bundle levels. When merging at the cash letter level, output x9 files will contain the complete cash letters from each of the input files. When merging at the bundle level, output x9 files will contain a single cash letter. Regardless of which option is selected, bundles are always copied in their entirety from input to output.
- A maximum file size must be defined for the output files to be created (default is 800MB). One or more output files will be created from each merge run. The command line provides the output x9 pattern name which is then suffixed with a sequential number for each file that is written by the merge.
- Input files which are successfully copied into the merged output can be optionally renamed to a new file extension on completion. This rename allows the file to be excluded from a future merge operation.
- Input files which are failed (they could not be copied into the merged output file) can be optionally renamed to a new file extension on completion. This rename allows the file to be clearly marked as failed for exception processing and will also exclude them from a future merge operation.
- Input files are typically only selected for the merge when they end in a type 99 file trailer, since the lack of a file trailer is a structural issue which means that either the file is in the process of being transmitted to the input folder, or that they are flawed. In either case, files without type 99 file trailers would always be excluded from a merge.
- Input files that were recently created can be further bypassed from one merge operation and thus deferred to the next scheduled run, with the intent to skip over files that may be actively transmitted. This skip interval can be set via the command line and defaults to 60 seconds.

- Output files are initially created as TEMP and are renamed to their final names on the ultimate completion of the merge. These renames are performed at the end of the merge, for all output files that were created by the merge run, and before the individual files are renamed to their merged extension.

Landing Zone Batch Script

Merge would typically be run from a batch script that would be periodically invoked by a scheduler. Part of this batch script would send an email if certain embedded functions are determined to be failed. The origination of this email can be done directly from Windows PowerShell but can also be done using tools such as SwithMail, CMail, SendSMTP, Blat, MailSend, SendEmail, or Send-It_Quietly. We would suggest writing a separate notification batch script that can be invoked with the failed message needs to be sent. The merge batch script would perform the following basic functions:

- Zip the landing zone contents to an external backup folder using an output file name that is fully date-time stamped. This zip file would be used on an exception basis for either research or to allow this merge operation to be completely rerun should that be needed. The backup is needed since it is assumed that the content of the landing zone will be a constant state of change hence a snapshot backup is required.
- Abort if this zip was unsuccessful.
- Run the merge using options to rename both merged and failed files.
- Abort if the exit status posted by merge is negative.
- Zip all files which were renamed to “merged” to a backup file in an external folder using a file name that is fully date-time stamped. This zip file is used for archive purposes and represents all files that were merged in this run.
- Abort if this zip was unsuccessful.
- Delete all files that were renamed to “merged” since they have now been written to the archive zip file.
- Abort if this delete was unsuccessful.
- Send an email if the merge exit status indicates that the landing zone contains failed files. This will be true if the failed files either existed prior to the merge run or were actually created by this specific merge run.
- Otherwise all is successful.

Landing Zone Watcher Script

Merge can have an optional watcher script that can be scheduled to monitor the merge process. The purpose of the watcher script is to be run independently and to ensure that the merge is running periodically as expected. The “utsf” merge option can be used to update a time stamp file within the landing zone for each independently scheduled run of merge (this update is one of the very last steps of the merge processing run). Because of that, the last modified date-time of the time stamp file is a good indication that a merge has completed. Suppose that merge is running on the hour. If that is the case, then the watcher task can be run on the half hour and can send an email alert if the merge time stamp file has not been updated as expected (for example, the time stamp is more than 60 minutes old).

Command line options

Switch	Description
-exti:”x1|x2|x3|...”	Provides a list of one or more file extensions that are used to identify x9 files within the input folder to be selected for merge. Files that do not match these extensions will be bypassed and not selected. Extensions are separated using the pipe character. Usage examples are -exti:x9 and -exti:“x9|x937”. Note that quote marks are needed around this parameter due to the embedded “|” special character.
-extr:merged	Indicates the file extension to be used for processed file renames. Processed files will not be renamed when no extension is provided.
-extf:failed	Indicates the file extension to be used for failed file renames. Failed files will not be renamed when no extension is provided.
-max:nnnn	Controls the maximum file size for each output file in MB. When this facility is activated, each output file will be named using a suffix beginning at 1 (outputFile_1.x9, outputFile_2.x9, etc). Default maximum file size (when not specified) is 800MB. This limit can be set to zero for unlimited file size, and in that situation file suffixes are not assigned. A value of “900mb” would be provided to override the maximum file size to 900mb. This same value could alternatively be specified as “921600kb”. When the maximum limit is being controlled based on items (and not bytes), the maximum might be assigned as “10000” or “20000”.
-creditsAddToItemCount:true	Indicates that credits are added to trailer item counts. The value can be assigned as either true or false.
-creditsAddToItemAmount:true	Indicates that credits are added to trailer item amounts. The value can be assigned as either true or false.
-creditsAddtoImageCount:true	Indicates that credits are added to trailer image counts. The value can be assigned as either true or false.
-mrgb	Indicates that the merge should be run at the bundle level, which means that the output will contain a single cash letter. Default is false.
-modb	Indicates that the bundle origination and destination routing numbers should be modified to match the cash letter header, as required by various x9 specifications. Default is false.
-gbic	Indicates that the selected input files should be grouped by item count instead of the default, which is by file size.
-sf	Indicates that subfolders (within the input folder) should be included.
-sd	Indicates that the selected input files should be sorted descending

Switch	Description
	on their control factor (size or items). This can reduce the number of output files when a large number of files are being created, since it increases the potential to group more files during the merge.
-skpi:nnn	Indicates a number of seconds that is compared against each individual file creation time and is used to bypass files that are very recently created within the input folder and may be in the process of being transmission. Files that do not meet this minimum skip interval are considered as “in-progress” and will be bypassed. Default value is 60.
-utsf:fileName.csv	Indicates that the optional time stamp file should be created. The time stamp file name will be defaulted to mergeTimeStamp.csv when not provided via the switch setting. If a time stamp file name is provided, it can be a base name or a fully qualified file name. If only a base name is provided, then the file will be created in the output folder.
-t99m	Indicates that input files should be selected even when they do not contain a type 99 file control trailer. This option would only be enabled in very unusual merge situations.
-dnr	Indicates that file renames are not to be performed on completion. This switch is help helpful during testing since a merge test can be run repetitively without the inputs being changed.
-config:	Use a specific x9 configuration which defaults to “x9.37”. For example, the CPA 015 standard can be selected as: “-config:x9.CPA_015”.
-j	A summary JSON file will be created with a suffix of “_summary.json” in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of “_summary.txt” in the same folder as the input x9 file(s).
-x	A summary XML file will be created with a suffix of “_summary.xml” in the same folder as the input x9 file(s).
-l	Will list all x9 records to the system log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -merge -exti:x9 <inputFolder> <output.x937>

merges files within extension x9 within <inputFolder> to output files <output_#.937> with all defaults applied and with no renames.

x9util -merge -exti:x9 -max 1400 <inputFolder> <output.x937>

merges files with extension x9 within <inputFolder> to output files <output_#.937> with a maximum output file size of 1400MB and with no renames.

x9util -merge -j -exti:x9 -max 1400 -extr:merged -extf:failed <inputFolder> <output.x937>

merges files with extension x9 within <inputFolder> to output files <output_#.937> with a maximum output file size of 1400MB. Merged files will be renamed to a new "merged" extension and failed files will be renamed to a new "failed" extension. A summary JSON file will be created.

x9util -merge -exti:x9 -max 1400 -extr:merged -extf:failed -skpi:120 -utsf <inputFolder> <output.x937>

merges files with extension x9 within <inputFolder> to output files <output_#.937> with a maximum output file size of 1400MB. Merged files will be renamed to a new "merged" extension and failed files will be renamed to a new "failed" extension. Input files must exist for a minimum of 120 seconds to be eligible for the merge. A time stamp file will be created in the input folder to allow a watcher task to monitor executions.

Time Stamp file format

The time stamp file can be optionally created by providing the "-utsf" switch. This file is designed to fulfill multiple purposes:

- A production process can monitor the last-modified date of the time stamp file (perhaps from a scheduled task) to ensure that a continuous process is running as expected, and issue alerts if the time stamp file becomes stale.
- Your application can extract information from the time stamp file for your own specific purposes. It is written in CSV format to help facilitate that.

The time stamp file contains the actual time stamp value, an exit message, counters for this merge operation, and detail lines which define each input file and the corresponding output it was written to. The format is as follows:

Line 1 contains: "timeStamp", time stamp.

Line 2 contains: "exitMessage", exit message.

Line 3 contains: "statistics", exit status, merge count, rename count, successful file count, failed file count, debit count, debit Amount, credit count, credit amount, cash letter count, and bundle count.

Next lines contain information about failed files; this list will indicate failed,!!none!! when there are no failed files; otherwise the format is: “failed”, failed file name, failed reason

From this point forward, there will be a list of the output files that have been written, along with a mapping of the input files that have been written to each of those files, where each of these lines are formatted as follows: “output”,output file number, output file name, input file name, input file length, record count, debit count , debit amount, credit count, credit amount, bundle count, micr valid amount, admin count , pre-note count, debit image count, credit image count, debit reversal count, debit reversal amount, credit reversal count, credit reversal amount, addenda count, cash letter count, hash total.

Exit status

Merge will set the overall run exit status as follows:

- -3 = input folder not found
- -2 = invalid function
- -1 = aborted
- 0 = run successfully with no errors
- 1 = run successfully with multiple files created
- 2 = run successfully with one or more failed files that exist in the input folder

Compare

Compare is designed for regression test purposes to confirm that two x9 files are logically equal. The content of both data and images will be compared. Compare has several benefits over a simple binary compare of the two files:

- Compare requires that the items within the file be structured in the same logical order to allow the matching process to remain aligned within the two files. Positioning is internally maintained using record types and then additionally on the item amounts and item sequence numbers. There are no other assumptions or requirements for the matching process. Any missing or inserted items will be reported and optionally listed.
- Within each item, addenda records are compared and differences are reported. The number of addenda records can be different and the compare will continue.
- Individual fields can be excluded from the comparison when their content is expected to be different.
- As part of the comparison, text files are created which contain the actual data records (excluding images) from the two files. These text files can be subsequently analyzed using your choice of DIFF tools.

Command Line Options

Switch	Description
-exclude	Provides a list of one or more fields which are to be excluded from the comparison. Usage examples are -exclude:"1.6|1.7|10.7". Note that quote marks are needed around this parameter due to the embedded "|" special character.
-delete	Deletes the records1 and records2 text files on completion.
-v	Enables verbose mode which will list inserted and/or deleted records into the comparison results.
-mask	Enables masking of excluded fields in the text files which are created.
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific "-config:" value. For example, the CPA 015 standard can be selected as: "-config:x9.CPA_015".
-j	A summary JSON file will be created with a suffix of "_summary.json" in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of "_summary.txt" in the same folder as the primary x9 file and summarizes its content.
-x	A summary XML file will be created with a suffix of "_summary.xml" in the same folder as the primary x9 file and summarizes its content.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.

Switch	Description
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -compare -exclude:”1.6|1.7|10.7” -delete <primary.x937> <secondary.x937>

compares primary.x937 against secondary.x937 and creates output txt and csv files within the primary file folder. Several fields are excluded from the comparison. The records1 and records2 text files will be automatically deleted on completion.

x9util -compare -exclude:”1.6|1.7|10.7” -delete <primary.x937> <secondary.x937> <results.csv>

compares primary.x937 against secondary.x937 and creates output results.txt and results.csv files (both located in the results file folder). Several fields are excluded from the comparison. The records1 and records2 text files will be automatically deleted on completion.

x9util -compare -j -exclude:”1.6|1.7|10.7” -delete <primary.x937> <secondary.x937> <output.txt> <results.csv>

compares primary.x937 against secondary.x937 and creates output.txt and results.csv files. Several fields are excluded from the comparison. The records1 and records2 text files will be automatically deleted on completion. A summary JSON file will be created.

Exit status

Compare will set the overall run exit status as follows:

- -2 = invalid function
- -1 = aborted
- 0 = run successfully with no differences
- 1 = run successfully with unmasked differences

Scrub

Scrub reads an x9 file in industry defined formats and applies user defined actions to remove proprietary and confidential information from the input file creating a new output file.
In addition to the data fields, scrub can optionally apply replacement actions against the images associated with each item. The actions to be performed can be saved in a user defined scrub configuration file which is available for future reuse. Scrub has a standard list of available cleansing actions that can be selected based on your specific requirements.

Scrub uses an xml parameter file to define the actions to be applied to the input file. This xml file is created and testing using our X9Assist product where you can define and review these parameters and repetitively test them against x9 files to ensure that they are providing desired results. These xml definitions are stored by X9Assist in folder / Documents / x9_assist / xml / scrub /. X9Assist allows you to create as many of these definitions as you need to perform your desired scrub actions. The shared used of these xml parameter files between X9Assist and X9Utilities is a key part of our design of these tools.

Command line options

Switch	Description
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific “-config:” value. For example, the CPA 015 standard can be selected as: “-config:x9.CPA_015”.
-exto:	Defines the output extension to be assigned to the created output x9 file when that file name is not explicitly specified on the command line; default is “new”.
-j	A summary JSON file will be created with a suffix of “_summary.json” in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of “_summary.txt” in the same folder as the input x9 file.
-x	A summary XML file will be created with a suffix of “_summary.xml” in the same folder as the input x9 file.
-l	Will list all x9 records to the system log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -scrub <input.x937> <parameters.xml>

scrubs <input.x937> using options specified in <parameters.xml> and creates x9 file <input.new> and <input.csv> within the designated input folder.

x9util -x -scrub <input.x937> <parameters.xml> <output.x937 >

scrubs <input.x937> using options specified in <parameters.xml> and creates x9 file <output.x937 > and <output.csv> within the designated output folder. A summary XML file will be created.

x9util -j -scrub <input.x937> <parameters.xml> <output.x937 >

scrubs <input.x937> using options specified in <parameters.xml> and creates x9 file <output.x937 > and <output.csv> within the designated output folder. A summary JSON file will be created.

Scrub results CSV File

Scrub creates an output CSV file which defines the input and output x9 files and then identifies each field which has been scrubbed along with a counter of actions applied for that field. A sample of the results file is as follows:

"c:\\users\\x9ware5\\documents\\x9_assist\\files_Utilities\\Test file with 2 checks.x9","c:\\users\\x9ware5\\documents\\x9_assist\\files_Utilities\\Test file with 2 checks_TMP.new"
"Field 01.10 Immediate Origin name",1
"Field 01.11 File ID Modifier",1
"Field 01.6 File Creation Date",1
"Field 01.9 Immediate Destination name",1
"Field 10.11 Originator Contact Name",1
"Field 10.5 Cash Letter Business Date",1
"Field 10.6 Cash Letter Creation Date",1
"Field 20.5 Bundle Business Date",1
"Field 20.6 Bundle Creation Date",1
"Field 25.6 MICR On-Us",2
"Field 25.7 Amount",2
"Field 25.8 ECE Institution Item Sequence Number",2
"Field 26.4 BOFD Business (Endorsement) Date",2
"Field 26.5 BOFD Item Sequence Number",2
"Field 50.3 Image Creator Routing Number",4
"Field 50.4 Image Creator Date",4
"Field 52.19 image scrubs",4
"Field 52.3 Bundle Business Date",4
"Field 52.5 ECE Institution Item Sequence Number",4
"Field 90.6 ECE Institution Name",1
"Record type 70 trailers",1
"Record type 90 trailers",1
"Record type 99 trailers",1
"end"

ImagePull

ImagePull is designed to extract images from a series of x9 files driven by a user supplied pick list which identifies the specific items to be pulled. This function is designed to allow a large number of images to be pulled in a single run across a large number of x9 files. Items are identified using an input CSV file which is provided on the command line. Extracted images are written to an output image folder in the same byte format as they are present on the input x9 file. There is no attempt to either validate or transform the images. By default, front images will be written for each selected item. Back side images can be enabled using a command line switch.

Input CSV

ImagePull reorganizes a user provided pick list by x9 file and then dispatches the image extraction process to a series of background threads which are run in parallel. Pull requests can be either specific or generic:

- Specific pull requests must include the item sequence number and can then also optionally include the item amount and item date.
- Generic pull requests identify items using date, routing, account, and serial number.

In support of optimization, each x9 file will be read only once. When specific items are being pulled, the reading process for any single file will be terminated when all requests for that individual file have been satisfied.

The item picklist is a CSV which is formatted with the following columns:

Column	Presence	Content	Example
1	Mandatory	Fully qualified x9 file name enclosed in quotes and always using forward slash as file separators, enclosed within quote marks.	“c:/user/folder/fileName”
2	Optional	Item sequence number for the item to be pulled as identified on the type 25 (check) or type 31 (return) item.	44000001
3	Optional	Item amount as a further qualifier for the item to be pulled (specified as numeric and without leading zeroes.	10000
4	Optional	Date for items to be pulled; this date is obtained from the capture date in the type 50 front side image record.	20180620
5	Optional	Account number for items to be pulled; this number is obtained from the MICR OnUs field.	123456789

Column	Presence	Content	Example
6	Optional	Check serial number for items to be pulled; this number is obtained from either the OnUs or AuxOnUs fields.	123456789

Results CSV

A results CSV is created which identifies the images which have been extracted for each pull request. The results CSV includes the fully qualified file name for each image as well as various fields which are associated with the item group. The results CSV is written in the same sequence as the input picklist, regardless of how the items had to be internally reorganized for the pull process. The results CSV (by default) contains the following fields:

Column	Field Name	Content
1	x9fileName	X9 file name as echoed from the pull request.
2	itemSequenceNumber	Item sequence number as echoed from the pull request.
3	frontImage	Fully qualified name of the extracted front image file.
4	backImage	Fully qualified name of the extracted back image file.
5	recordType	Record type for the selected item group.
6	RecordNumber	Record number of the selected item group within the x9 file.
7	auxOnUs	Auxiliary OnUs field from the item record.
8	epc	External processing code from the item record.
9	payorRouting	Payor routing from the item record.
10	payorRoutingCheckDigit	Payor routing check digit from the item record.
11	onus	OnUs from the item record.
12	amount	Amount from the item record.
13	bofdIndicator	BOFD indicator from the item record.
14	returnLocationRouting	Return location routing number from the bundle header record.
15	bofdDate	BOFD from the first BOFD addendum record.
16	bofdRouting	BOFD routing from the first BOFD addendum record.
17	imageCreatorRouting	Image creator date from the first type 50 record.
18	imageCreatorDate	Image creator date from the first type 50 record.
19	returnReason	Return reason from the type 31 return record.

Extracted Image File Names

ImagePull will format the file name for each extracted images into a standardized and fully qualified format that includes the complete file system path. In certain error situations, the image file name is replaced by a constant string to identify a specific error condition as follows:

Error Image File Name	Error Condition
`"file-not-found"`	The x9 file associated with this picklist item was not found.
`"item-not-found"`	The item was not found based on the provided criteria within the identified x9 file.
`"item-no-image"`	The item was found but did not have an attached image.
`"file-aborted"`	An internal error occurred during the processing of the identified x9 file and the image is not available. More information on the error can be found in the created system log.

Error CSV

An error CSV is created for each ImagePull run which identifies those items which could not be pulled. The error CSV is always created and will be empty when there are no errors for a given processing run. The error CSV is created in the output CSV folder with the same named appended with "_ERRORS". This file can be used to analyze why images were not pulled for specific items and can be used as input for a subsequent ImagePull run should those error conditions be corrected. The format of the error CSV file is as follows:

Column	Field Name	Content
1	`x9fileName`	X9 file name as echoed from the pull request.
2	`itemSequenceNumber`	Item sequence number as echoed from the pull request.
3	`amount`	Amount from the item record.
4	`CSV input line number`	Line number of this request on the input CSV request file.
5	`error condition`	Description of error encountered for this error request.

XML parameter file

An XML parameter file can be optionally provided to control the fields that are written to the output CSV file. When not provided, the output CSV will be created in our standard format with all available fields. The XML parameter file can be used to limit the fields that are included or to change the sequence of those fields. An example XML parameter definition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<imagePull>
 <copyright>X9Ware LLC 2013 2014</copyright>
 <product>X9Assist</product>
 <release>R4.01</release>
```

```
 <build>2019.07.24</build>
  <fields>
     <name>x9fileName</name>
     <name>itemSequenceNumber</name>
     <name>amount</name>
     <name>frontImage</name>
     <name>backImage</name>
  </fields>
</imagePull>
```

Command line options

Switch	Description
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific "-config:" value. For example, the CPA 015 standard can be selected as: "-config:x9.CPA_015".
-cr	Include selection of images for record credit record type 61 and 62.
-ib	Pull and write back side images; the default is to omit back side images.
-ix	A time stamp (date and time) is appended to the output image folder name to make it unique for each image pull run.
-ic	The output image folder for this run will be cleared when it already exists. This process will only remove TIFF files since those are the only expected file extensions which would be expected to exist in the output image folder.
-ia	Override to forcibly not abort when the output image folder is not empty. If you do not set this override switch, ImagePull will abort if the output image folder exists and contains one or more files (regardless of extension). The default action is to abort. This abort situation can be avoided through use of either the "ix" switch to append a time stamp to the image folder name (making it unique) or the "if" switch to clear the output image folder when it is determined to exist.
-l	Will list all type 25 and type 31 x9 records to the system log.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -imagePull <input.csv>

pulls images using the picklist as defined in <input.csv>, creates output CSV <input_RESULTS.csv> within the designated input folder, and writes images to <input_IMAGES> within the designated input folder.

x9util -imagePull <input.csv> <results.csv>

pulls images using the picklist as defined in <input.csv>, creates output <results.csv>, and writes images to <input_IMAGES> within the designated input folder.

x9util -imagePull <input.csv> <results.csv> <imageFolder>

pulls images using the picklist as defined in <input.csv>, creates output <results.csv>, and writes images to <imageFolder>.

x9util - imagePull <input.csv> <parameters.xml> <results.csv> <imageFolder>

pulls images using the picklist as defined in <input.csv> including the specific fields as defined in <parameters.xml>. Creates <results.csv> with images written to <imageFolder>. All four (4) command line files are required to prevent ambiguity. The parameter xml identifies the fields to be included in the output CSV and the order that those fields will be written. Fields are identified by name and must be selected from a list of system available field names. Other command line formats (which do not include the parameters xml file) will otherwise include all available output fields in a predetermined order.

Update

Update reads an existing x9 file in common industry defined formats and performs replacement operations against one or more fields within the file. This process consists of searching for user defined values within specific record types and fields and then replacing those values (when found) with new data values in a newly created x9 file.

The match/replace data is provided in a user supplied update file (xml format) which is provided via the command line.

Command line options

Switch	Description
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific "-config:" value. For example, the CPA 015 standard can be selected as: "-config:x9.CPA_015".
-outExtension:	The default x9 output file extension to be used which defaults to "new".
-j	A summary JSON file will be created with a suffix of "_summary.json" in the same folder as the output x9 file.
-t	A summary text file will be created with a suffix of "_summary.txt" in the same folder as the input x9 file.
-x	A summary XML file will be created with a suffix of "_summary.xml" in the same folder as the input x9 file.
-l	Will list all x9 records to the system log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -update <input.x937> <parameters.xml>

reads <input.x937> using update parameters defined in <parameters.xml> and creates x9 output file <input.new> and results file <input.csv> within the designated input folder.

x9util -update <input.x937> <parameters.xml> <output.x937>

reads <input.x937> using update parameters defined in <parameters.xml > and creates output files <output.x937> and results file <output.csv> within the designated output folder.

x9util -update <input.x937> <parameters.xml> <output.x937> <results.csv>

reads <input.x937> using update parameters defined in <parameters.xml> and creates output file <output.x937> and results file <results.csv>.

x9util -j -update <input.x937> <parameters.xml> <output.x937> <results.csv>

reads <input.x937> using update parameters defined in <parameters.xml> and creates output file <output.x937> and results file <results.csv>. A summary JSON file will be created.

x9util -x -update <input.x937> <parameters.xml> <output.x937> <results.csv>

reads <input.x937> using update parameters defined in <parameters.xml> and creates output file <output.x937> and results file <results.csv>. A summary XML file will be created.

Update Results CSV File

Update creates a results CSV file that has one line for each field that is modified, serving as an audit trail for the modifications that have been made. The columns for this file are as follows:

Column	Content
1	Record number.
2	Record dot field that was updated.
3	Field name.
4	Old value
5	New value.

Constants

The update xml deck can include constants that are defined once and then repetitively reused. These are defined in an optional “constants” section. Each constant is defined with a user defined name, beginning and ending with %, and with an associated value that can subsequently be used on a <replace>. For example, various records commonly use ECE Institution Routing as a logical value. An example is as follows:

```
<constants>
        <id>%ece-institution%</id> <value>123456780</value>
</constants>
```

Look Back to Previous Values

The update xml deck can assign the value to the current field based on values that were assigned to an earlier record. For example, it might be desirable to assign field 52.2 (ECE Institution Routing Number) with the same value as 10.4 (ECE Institution Routing Number). Look backs for an earlier record type final value for that field (after possible updates are applied). Look backs for the

current record type assign the input value for that field, despite the fact that value may be subsequently altered. Look backs are done using the record and field number. For example, the string identifier for an input field look back for 10.4 would be 10.4, while the output field look back would be 10.4. These identifiers are utilized within <replace>.

External Table Lookups

There are situations where a field needs to be updated using data that does not exist within the current file. An example of this would be updating the payor deposit account number in fields 26.6 or 32.6, where the value needs to be assigned based on a lookup that is based on item sequence number and item amount.

This type of lookup is applied utilizing a table that must be constructed by a user application. Following the above example, the table might contain item sequence number and amount, to provide the actual deposit account number. The table might appear as follows:

```
44000001/10002 88441
44000002/10004 88442
44000003/10006 88443
```

The table reference must include table name and one or more fields that are used to build the lookup key. The table can be referenced through either an absolute (fully qualified) or relative file name. If the table name is relative, then it must be defined in the same folder as the update xml file itself. Following the above example, the <replace> value might be:

```
<replace>//table/accountLookupTable.txt/$32.5$/$31.5$
```

Update XML File Examples

The update xml deck allows specific fields within the input file to be inspected for one or more values, which can then be replaced with a new target value. When searching for multiple values, all existing input values can be mapped to the same or different replacement values.

The x9 rules unfortunately define separate routing and routing check digit fields for the payor routing in record types 25 and 31. To facilitate replacement of those fields are considered combined as a single nine-digit field for both the old and new values. This concatenation of data applies only to those two specific situations.

Update contains <match> and <replace> parameters at the field level.

The <match> parameter supports a variety of search functions that can be used to identify the values to be replaced. These are as follows:

- <match/> matches against blank values.
- <match>*</match> is a wild card that matches against any and all values.
- <match>=xx</match> searches for a specific value (in this case, xx).
- <match>=xx|yy|zz</match> searches for a list of specific values (in this case, xx, yy, zz).

- <match>regexString</match> matches against a user provided Java RegEx string. This is a more advanced capability but also provides the most powerful matching capabilities.

The <replace> parameter provides the immediate replacement value when searching for specific values. However, in the case of searches that are based on RegEx, the <replace> parameter instead provides the replacement value for that portion of the input value that was successfully matched by RegEx. Essentially, you need to picture the input value as containing three separate parts. First is a series of leading characters, following by the portion that is matched by the RegEx string, and then finally a series of trailing characters that follow the RegEx matched portion of the string. The matched portion of the input stream will be replaced with the <replace> value. The leading and trailing characters will be retained.

The XML file defines the fields to be queried, the match rules to be applied, and the replacement value that will be assigned. An example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<update>
  <constants/>
 <swaps>
  <swap>
        <field>1.9</field> <values> <match>*</match> <replace>Central</replace>
</values>
  </swap>
  <swap>
        <field>1.10</field> <values> <match>*</match> <replace>Central</replace>
</values>
  </swap>
 </swaps>
</update>
```

RegEx Online Tools

There are numerous Java RegEx online tools that can be used to develop your development of a Java RegEx regular expression. These tools allow you to enter your expression and a corresponding data string to test against. This interactive capability is invaluable during this work. An example site is: https://regex101.com/ . When using these sites, make sure you identify the HTML “flavor” being used as Java8. You will want to enter various input data to determine that matching is working as expected. Also include some negative testing as well, to make sure that other arbitrary data is not accepted as a match. One thing nice about regex101.com is that it also generates a description of what and how the RegEx string operates.

AI Assistance

Chat-GPT (and other similar AI tools) can be used to develop your RegEx strings. These tools are great because you can describe your requirements in a great level of detail, and the tool will then automatically build the RegEx string per your stated needs. You need to make sure you mention that you are using Java HTML strings, since various programming environments can slightly vary.

Once you get a RegEx string from your AI tool, you must then take that to an online test site (as mentioned above) for your detailed testing. At this point, you will want to include both positive and negative test cases. An important point to remember is that the AI tool make format your RegEx strings such that a single backslash character '\' is actually escaped and thus provided as '\\'. This means that you must enter change those '\\' sequences to just '\' in your online testing, and also when you enter the RegEx string into your XML file.

RegEx Examples

The following RegEx match-replace examples may help:

Input Value	Match	Replace	Output Value	Comments
123456789/22	/[0-9]{2,3}$	/33	123456789/33	This match string looks for a two or three digit number at the end of the string and replaces that with "33" when present. This is an example of a two digit number being replaced.
123456789/222	/[0-9]{2,3}$	/33	123456789/33	This is an example of a three digit number being replaced.
123456789/2222	/[0-9]{2,3}$	/33	123456789/2222	This is an example of a four digit number not being replaced.
123456789/	/[0-9]{2,3}$	/33	123456789/	This is an example of no trailing number whatsoever, and it not being replaced.
00012345	^0*		12345	The ^ anchors to the start of the string, and then matches 0* zero or more 0 characters . These are then replaced with the empty string to remove them.
12345	^0*		12345	This is an example of trying to remove leading zeroes when they do not exist in the input. Hence the string is unchanged.

Split

Split reads an existing x9 file in common industry defined formats and splits (divides) into output segments, where each output is a separate x9 file. The file, cash letter, and bundle headers are copied from the input x9 to the output x9. We will refer to each output x9 file as a logical segment.

- Each output segment (file) will always contain a file header and file trailer.
- Each output segment receives one or more items, as defined by the split xml parameter file (which is defined below).
- A skip segment can be created for items that are to be purposefully skipped.
- A default segment can be optionally created as a catch-all for all items that were otherwise unselected by all defined criteria.
- A given cash letter header/trailer is only copied when that cash letter contains items.
- A given bundle header/trailer is only copied when that bundle contains items.
- All trailer records (bundles, cash letters, and file) will be automatically recomputed.

Items are routed to output segments based on a logical split key, which is defined based on fields that must be located within:

- The file header record.
- The cash letter header record.
- The bundle header record.
- The individual item.
- The primary addenda (endorsement) record that is attached to the item (type 26 or 33).

Each output segment then receives items that represent one or more split keys. For example, you could use the facility to define output segments based on:

- The file header destination routing.
- The file header and cash letter header destination routing.
- The item routing.
- The item MICR OnUs transaction code.
- Subsets of any of these fields; for example, selective substring digits from the type 26/33 item sequence number.

Command line options

Switch	Description
-config:	Use a specific x9 configuration. When this parameters is omitted, the file header will be inspected to determine the most appropriate x9 configuration to be used. However, you may also provide a specific “-config:” value. For example, the CPA 015 standard can be selected as: “-config:x9.CPA_015”.
-j	A summary JSON file will be created with a suffix of “_summary.json” in the same folder as the output x9 file.

-t	A summary text file will be created with a suffix of "_summary.txt" in the same folder as the input x9 file.
-x	A summary XML file will be created with a suffix of "_summary.xml" in the same folder as the input x9 file.
-l	Will list all x9 records to the system log.
-batch	Invokes batch (folder based processing; see that earlier topic for more information.
-workUnit:	Assigns command line parameters and files from an xml file that was previously created and saved by the X9Utilities facility within X9Assist. This allows you to easily repeat an X9Assist task directly in X9Utilities batch.

Command line examples

x9util -split <input.x937> <parameters.xml>

reads <input.x937> using split parameters defined in <parameters.xml> and creates x9 segments (named per xml content) with a corresponding results file <input.csv> within the designated input folder.

x9util -split <input.x937> <parameters.xml> <outputFolder>

reads <input.x937> using split parameters defined in <parameters.xml > and creates segments (named per xml content) with a corresponding results file <input.csv>. Named output files can have relative file names and will be created in the designated output folder.

x9util -split <input.x937> <parameters.xml> <outputFolder> <results.csv>

reads <input.x937> using split parameters defined in <parameters.xml> and creates segments (named per xml content) with a corresponding results file <results.csv>. Named output files can have relative file names and will be created in the designated output folder.

x9util -j -split <input.x937> <parameters.xml> <outputFolder> <results.csv>

reads <input.x937> using split parameters defined in <parameters.xml> and creates segments (named per xml content) with a corresponding results file <results.csv>. Named output files can have relative file names and will be created in the designated output folder. A summary JSON file will be created.

x9util -x -split <input.x937> <parameters.xml> <outputFolder> <results.csv>

reads <input.x937> using split parameters defined in <parameters.xml> and creates segments (named per xml content) with a corresponding results file <results.csv>. Named output files can have relative file names and will be created in the designated output folder. A summary XML file will be created.

Default Output Segment

The default output segment is optional and serves as a catch-all for any items that were not selected and written to any other output segment. The default segment can be used as an exception file to identify unexpected items based on the segment selection criteria. Exit code four will be posted when the default output segment contains items.

Skipped Items

In addition to default items, there is also the potential for skipped items. This facility is very useful to allow you to skip credits, since split is most typically targets debits. Skipped items will not be written to any output segment.

Auto-Reconcilement

Split accumulates the number of items written to all output segments (plus the skipped items) and will abort if that item count does not match the number of input items.

Output Segment Totals

Split accumulates debit and credit totals for each output segment and includes those totals in the system log.

Output Segment File Names

Output file names can be defined in a variety of ways:

- The base file name (base+extension) can be defined within the XML file, with the output folder then defined as a command line parameter. This approach allows the same XML file to be used for alternative purposes (for example, test versus production).
- The base file name (base+extension) can be defined within the XML file, without having an output folder defined as a command line parameter. In this situation, the output folder will be defaulted to the same folder where the input file resides. This approach allows the same XML file to be used for alternative purposes (for example, test versus production).
- Finally, the XML file can define a fully qualified output file name (folder plus base file name including extension. When this approach is used, the file name is completely self-defining, with no further parameters needed from the command line.

Split Results CSV File

Split creates a results CSV file that has one line for each output segment created. The columns for this file are as follows:

Column	Content
1	Output file name.
2	Record count.
3	Debit count.
4	Debit amount.

5	Credit count.
6	Credit amount

Split XML Tag Names

The following basic options are available in the split xml file:

Xml Tag Name	Content	Default
<defaultFileName>	Default output file to be written, for those items that are unselected by all criteria.	Omitted.
<dateTimeStamp>	A date pattern to be utilized when output segments are to be suffixed with a time stamp. This facility can be used to make all output file names unique. A commonly used assignment would be: yyyyMMdd_HHmmss . You can do an internet search on "Java Date Format Pattern" for all allowable pattern characters and usage examples.	Omitted.
<doNotRewrite>	Indicates if an existing output segment can be overwritten. Values can be true or false.	False.
<debitsOnly>	Indicates that only debits should be selected. Values can be true or false.	False.
<creditsOnly>	Indicates that only credits should be selected. Values can be true or false.	False.

Split XML file examples

The update xml deck allows specific fields within the input file to be inspected for one or more values, which can then be replaced with a new target value. When searching for multiple values, all existing input values can be mapped to the same or different replacement values.

The x9 rules unfortunately define separate routing and routing check digit fields for the payor routing in record types 25 and 31. To facilitate replacement of those fields are considered combined as a single nine-digit field for both the old and new values. This concatenation of data applies only to those two specific situations.

Update contains <match> and <replace> parameters at the field level.

The <match> parameter supports a variety of search functions that can be used to identify the values to be used for the split operation, as follows:

- <match/> matches against blank values.
- <match>*</match> is a wild card that matches against any and all values.
- <match>=xx</match> searches for a specific value (in this case, xx).
- <match>=xx|yy|zz</match> searches for a list of specific values (in this case, xx, yy, zz).
- <match>=micr-account-list-xxxxxx|xxxxxx|xxxxxx|...</match> searches the MICR OnUs field (eg, field 25.6) against a list of one or more account numbers, separated by '|'. This

rule is provided because the doing this using RegEx is difficult, given the various content within the MICR OnUs field. A usage example would be: <match>=micr-account-list-11111|22222|33333|44444</match>.

- <match>=micr-account-table-[fullyQualifiedFileName]</match> searches the MICR OnUs field (eg, field 25.6) against an external text file that contains zero or more account numbers, where each account number if presented as a separate line within the file. If there are 500 account numbers, then there will be 500 lines within the text file. A usage example would be: <match>=micr-account-table-C:\Users\SomeUserID\Documents\accountLookupTable.txt</match>. Note that this file name is not enclosed in brackets.
- <match>regexString</match> matches against a user provided Java RegEx string. This is a more advanced capability but also provides the most powerful matching capabilities.

The <replace> parameter provides the immediate replacement value when searching for specific values. However, in the case of searches that are based on RegEx, the <replace> parameter instead provides the replacement value for that portion of the input value that was successfully matched by RegEx. Consider the input value as containing three separate parts. First, a series of leading characters, followed by the RegEx match string, and finally a series of trailing characters that follow the RegEx matched portion of the string. The matched portion of the input stream will be replaced with the <replace> value. The leading and trailing characters will be retained.

The XML file defines the fields to be queried, the match rules to be applied, and the replacement value that will be assigned. An example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<split>
  <outputs>
          <defaultFileName></defaultFileName>
          <debitsOnly>false</debitsOnly>
          <creditsOnly>false</creditsOnly>
          <output> <fileName>F111111118.x9</fileName> <writeEnabled>true</writeEnabled>
            <fields>
              <field>20.3</field> <values> <match>111111118</match> <replace/> </values>
                  </fields>
          </output>
          <output> <fileName>F222222226.x9</fileName>
            <fields>
              <field>20.3</field> <values> <match>222222226</match> <replace/> </values>
                  </fields>
          </output>
  </outputs>
</split>
```

RegEx examples

Refer to “update” for RegEx match-replace examples.

Embedded Use of the X9Ware SDK

X9Utilities is based on the X9Ware SDK. This foundation requires that certain folders be populated within your run time environment. These folders are installed as part of the X9Utilities package and are required for execution.

- /backup
- /images
- /invalidImages
- /properties
- /rules
- /xml

Do you need a tool with more capabilities than just x9 read and write? Please let us know; perhaps we can add to our X9Utilities product. You can do virtually any x9 related task using the X9Ware SDK, so it should always be considered when you have complex tasks to be performed.

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:

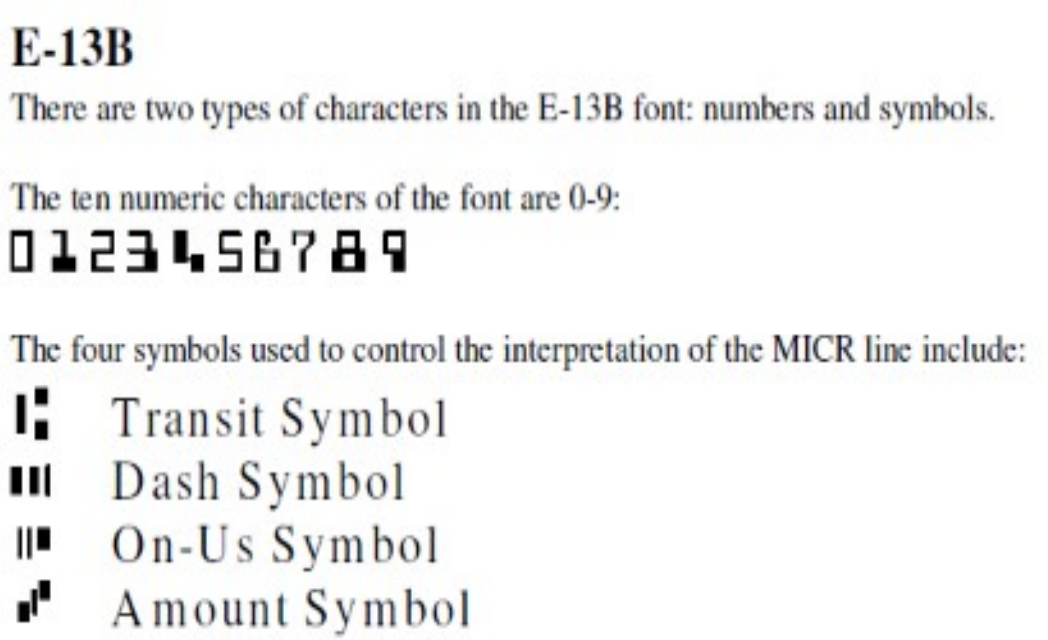
- Auxiliary On-Us
- Transit
- On-Us
- Amount

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



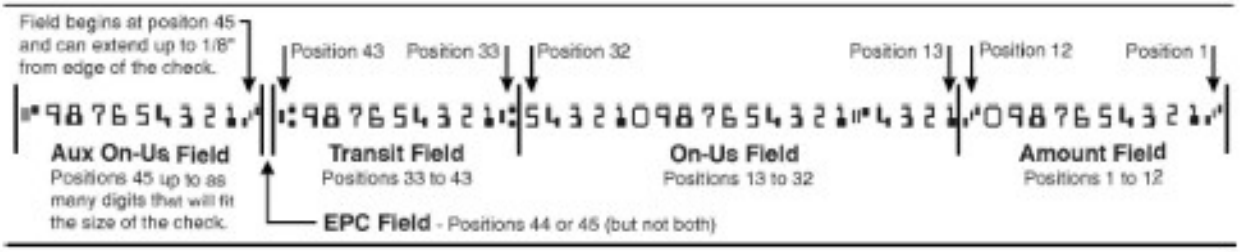
MICR Line Fields

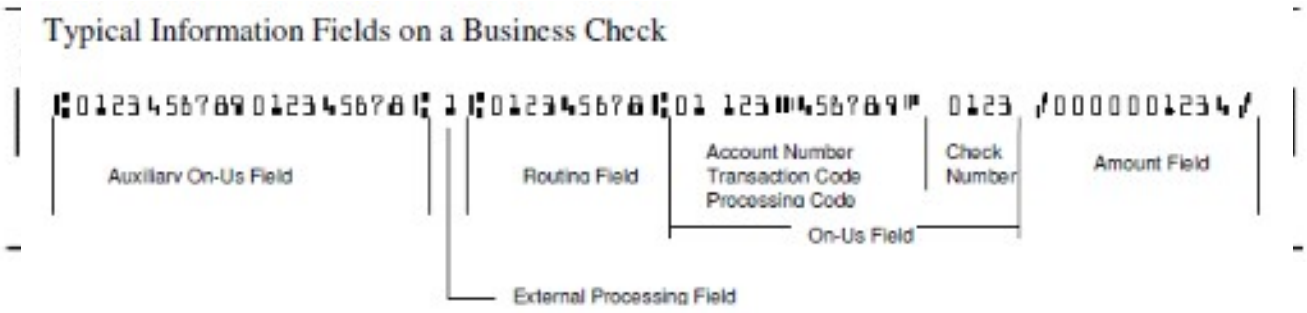
MICR line fields (from right to left on the check) are as follows:

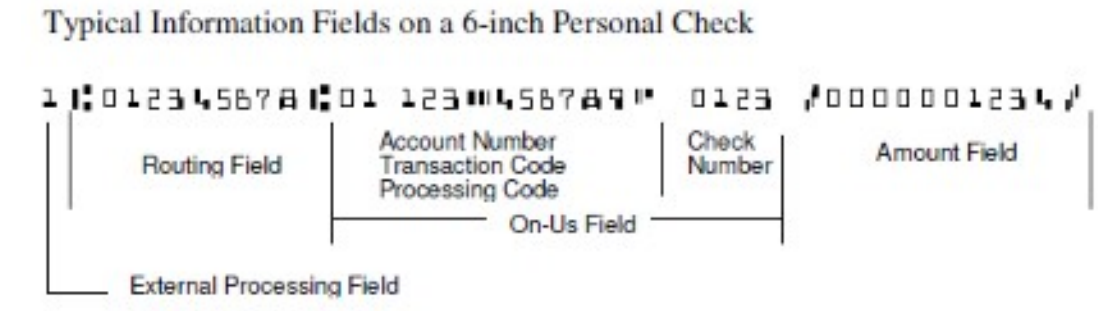
Field #	Field Name	MICR Positions	Description
1	Amount	1 – 12	Amount with leading and trailing E13B amount symbols. This field is typically not encoded in the image environment.

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 AAAAAAAAAAAAAAAAAAAAB,
- 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

Field	RegEx	Regex Notes
		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 B: 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 hieratchies. 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	Primarily used for

XML Group	XML Field Name	Populated Into	Notes
		52.19	documentation, but also inserted into the credit image when <creditImageDrawFront> is true. When using credit profiles, the account name 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,	Numeric; default is current

XML Group	XML Field Name	Populated Into	Notes
		10.06	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” 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	

XML Group	XML Field Name	Populated Into	Notes
<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 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	Default (when this field is omitted) is to create as “hhmmssSSS” which satisfies the common requirement that the cash letter identifier be unique for a given day. There are several other alternatives: • “xxxxxxxx” (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

XML Group	XML Field Name	Populated Into	Notes
			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 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". • "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

XML Group	XML Field Name	Populated Into	Notes
			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	
<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	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: • "xxxxxxxxxx" (up to 10 character string) which is directly assigned to all bundles. • "sequential" which assigns an incremented bundle identifier beginning with "0000000001".

XML Group	XML Field Name	Populated Into	Notes
			• "%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 first bundle. • "#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

XML Group	XML Field Name	Populated Into	Notes
			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 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

XML Group	XML Field Name	Populated Into	Notes
			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 • "metavante" – an industry standard type 61 credit with 13 fields that is defined in x9rules as format 61-001. • "dstu" – an industry standard type 61 credit with 12 fields that is defined in x9rules as format 61-002. • "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		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 rules (eg, "metavante") or on an absolute basis using the record format (eg, "61-001"). Various options (from those that follow) must be used to further configure the constructed credit. "creditInsertedAutomatically" must be enabled to activate this feature. "creditLocation" must be assigned to define where the generated credit will be inserted into the file. The default is that all credits will begin in a newly created bundle. Images can be either dynamically drawn or provided from external image files.

XML Group	XML Field Name	Populated Into	Notes
	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”.		Primary and secondary endorsements can be created and attached to the credit. 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.
<fields>	creditRecordLocation Supported values are: • 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		Identifies the location where the credit should be inserted into the created x9 file. 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. The most commonly used setting is “a20” which will insert the credit after the first bundle header record. A value of “none” must be used when a credit format of “t10” has been assigned, since there is no actual credit to be inserted.
<fields>	creditInsertedAutomatically This facility is used to insert one or more deposit tickets into the created file when required by the receiving bank. Content of the deposit ticket must be defined using the other credit xml fields below (creditPayorBankRouting, creditMicrOnUs, creditMicrAuxOnUs, creditItemSequenceNumber, etc). The deposit ticket can optionally contain		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.

XML Group	XML Field Name	Populated Into	Notes
	attached proxy images which are created using external tiff images which are identified by creditImageProxyFront and creditImageProxyBack.		
<fields>	creditStructure Provides further direction regarding the creation of individual credits. This parameter is applicable only when creditInsertedAutomatically has been enabled. Credit structure allows the checks with the deposit to be grouped in specific ways, subject to customer or financial institution requirements. 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.		“multiItem” creates a credit which is offset by multiple checks. This is the default value and represents standard processing. “singleItem” creates single item deposits (every check is created within its own deposit). “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.
<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

XML Group	XML Field Name	Populated Into	Notes
			when the [“credit”, amount] format is present on the items csv file.
<fields>	creditMicrAuxOnUs	6x.xx, 25.02	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: “auto” will assign a 10 digit number as yymmddhhmm. "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. “hhmmssnnnn” will assign a 10 digit number where nnnn is a sequential number that is incremented for each new credit. “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 the credit back to the attached items.
<fields>	creditItemSequenceNumber	6x.xx, 25.08	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: “auto” will assign a 15 digit number

XML Group	XML Field Name	Populated Into	Notes
			as yymmddhhmmssnnn where nnn is a sequential number that is incremented for each new credit. "yyjjjhhmmssnnn" will assign a 15 digit number as yyjjjdhhmmssnnnn where jjj is the Julian day within the current year and nnnn is a sequential number that is incremented for each new credit. "yyyymmddhhmmss" will assign a 14 digit number as yyyymmddhhmmss.
<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 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	Default is "false". Indicates that the

XML Group	XML Field Name	Populated Into	Notes
		Image	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 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,	Default to “false” since these counts

XML Group	XML Field Name	Populated Into	Notes
		90.04, 99.04	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 financial institution which has captured the image. This value is normally omitted and allowed to

XML Group	XML Field Name	Populated Into	Notes
			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 other way to accomplish this.

XML Group	XML Field Name	Populated Into	Notes
			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	28.xx	Third type 28 endorsement record. The populate date and populate

XML Group	XML Field Name	Populated Into	Notes
	prefix “secd3” instead of “secd”.		sequence number fields are not duplicated; 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	

XML Group	XML Field Name	Populated Into	Notes
<fields>	imageDataClippingCoordinateH2	52.11	
<fields>	imageDataClippingCoordinateV1	52.12	
<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 be more acceptable to receiving processors.
<fields>	micrTransitSymbol		Default is “A” and is not case sensitive; used to parse the MICR

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

Field	Field Name	Usage	Position	Length	Format	Notes
13	Addendum Count	M	77	2	N	Must be set to 00 when there are not addendums for this check detail 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)	C	13	8	N	

Field	Field Name	Usage	Position	Length	Format	Notes
	Date					
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	
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”.
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	

Field	Field Name	Usage	Position	Length	Format	Notes
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
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-Us 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".

Field	Field Name	Usage	Position	Length	Format	Notes
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	
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

Field	Field Name	Usage	Position	Length	Format	Notes
						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, 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

Field	Field Name	Usage	Position	Length	Format	Notes
						'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	C	75	1	AN	'0' Did not convert physical document

Field	Field Name	Usage	Position	Length	Format	Notes
	Indicator					‘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 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	
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
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	
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	
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	C	19	1	N	

Field	Field Name	Usage	Position	Length	Format	Notes
	(EPC)					
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	M	62	15	NB	

Field	Field Name	Usage	Position	Length	Format	Notes
	Number					
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

HeaderXml937 Editor

X9Vision ?	X9Validator ?	X9Assist?		X9.37 ?	ACH ?	CPA005 ?
NO	**YES**	**YES**		**YES**	**NO**	**NO**

HeaderXml937 Editor is an editor that allows the HeaderXml parameter files used by the X9Utilities "-write" command to be viewed, modified, and saved. These xml parameter files define the x9.37 attributes that are used by "-write" to generate output files.

HeaderXml937 Editor is available as part of our X9Validator and X9Assist desktop tools. These xml parameter files define the x9.37 attributes that are used by the X9Utilities "-write" function to generate output files.

This editor makes the process of creating and updating those parameters much easier than the alternative of using a simple text editor (eg, NotePad or NotePad++). Using our HeaderXml937 Editor eliminates the xml document errors that can often result from that alternative process. This editor allows you to concentrate on content and not all of the technical intricacies that are associated with an XML file.

There are a substantial number of fields defined within the HeaderXml definition, which can add to the complexity of both entering and updating these fields. To make things a bit easier, all fields have been grouped based on function, using tabs which are located in the right-most column of the editor. These groups bring related fields together onto a single panel. This makes it much easier to find any given parameter and allows you to easily view all of the parameters for that same topic.

Each of the field panels have several right-side columns that provide additional insight into the fields associated fields.

- The far-right column is formatted with [nn] where "nn" is the maximum length of this data field. The editor will begin to beep (as an error indicator) if you attempt to overfill the capability of any given field
- The second column from the right provided information as to how a given field is used, or where the entered data will be populated, which can be very useful information when creating and modifying these parameter files.

The following tabs are available within the editor:

- Structure – defines high level fields that are associated with the overall file.
- File Headers – defines fields that are associated with the type 01 record.

- Cash Letter Headers – defines fields that are associated with the type 10 record.
- Items – defines fields that are associated with the type 25 record.
- Credits – defines fields that are associated with the type 61/62/25 credit records.
- BOFD Addenda – defines fields that are associated with the type 26 (forward presentment) or type 32 (return) records that are attached to all items.
- Secondary Addenda – defines fields that are associated with the type 28 (forward presentment) or type 35 (return) records that are attached to all items.
- Image View Detail – defines fields that are associated with the type 50 record.
- Image View Data – defines fields that are associated with the type 52 record.
- MICR Symbols – defines the control characters that are to be used when building MICR line data from the underlying component fields.

The following functions are provided on the action line at the bottom of the editor panel:

- Cancel – exits the editor; anything that has been saved will remain in that state and will not be undone.
- Load – allows a new file to be loaded. The current content within the editor will be replaced with the chosen file; the current content will be lost.
- Save – allows the current editor contents to be saved to an output XML file.
- Reset – resets the editor to a default status.