



Fast Track Your Forms Processing Application Development

Getting data from paper forms into your database used to require a huge software development effort. Starting with the Accusoft Pegasus FormSuite SDK, however, you can accurately collect information from paper forms within days rather than months.

Getting Started

The first step is to [download the Accusoft Pegasus FormSuite SDK](#) (including sample code). The kit is available for both ActiveX and .NET development. We'll use C#.NET to show you how to quickly build a working forms processing application. The **FormSuite** download includes several components that do all the heavy lifting in forms processing:

- **ImagXpress**
The basic image processing component to open and manipulate document image files, including controls for displaying thumbnails and drawing zones on images
- **TwainPRO** and **ISIS Xpress**
Two components included with ImagXpress that let you easily control almost any scanner
- **FormFix**
Identifies which template matches a scanned form image, aligns that image to the template, drops out the blank form, extracts zones for recognition, and performs check-box recognition (OMR)
- **ScanFix**
Provides extensive image cleanup functions like deskew, despeckle, and others to improve the quality of scanned images
- **SmartZone OCR/ICR**
Recognizes the machine printed or hand-printed text in the fields passed by FormFix
- **FormDirector**
A traffic cop for easily passing data between the various forms processing components above
- **FormAssist**
A sample application showing you how to put all of the above pieces (except for the scanning part) together to build a complete, forms processing application

Our focus will be on the **FormAssist** application code, which is frequently used as the starting point for developers who are building their own forms processing systems. A tremendous amount of time can be saved by removing any unneeded features and

adding your own batch processing features, rather than starting with an empty project. The full source code for FormAssist is provided for exactly this purpose. All of the components used by FormAssist are installed by the FormSuite SDK installation. The entire application can be compiled and run with trial licensing—as long as it remains unlicensed reminder screens will pop up as various components are accessed.

Once you've installed FormSuite, navigate from your Start button to: All Programs | Pegasus Imaging | FormSuite | Samples | .NET | C# | FormAssist Demo. This link starts Visual Studio with the FormAssist project already loaded. FormAssist demonstrates two important but separate capabilities that your complete forms processing solution requires. First, it allows you to build a set of **form templates** that will be used to identify which form an incoming image matches, and how to process it. The image will automatically be scaled and aligned to the matching template form, even if it's upside-down. For each template form, you can define precisely where each variable field of data—or zone—is located, and how you want it to be processed.

The second part is the production process of running batches of scanned forms through the matching, alignment, extraction, and recognition steps. The scanning step, where you'll create the form images from your physical documents using TwainPRO or ISIS Xpress, and the actual delivery of the recognized data to your database, will not be addressed here. Excellent samples for document scanning are provided with those two components, and database access is highly specific to your particular needs.

Building a Form Template

So, let's create a new Form Set and add our first form template into it. To compile and run the project, all you need to do is Start Debugging (F5). After FormAssist starts, click File | New Form Set:

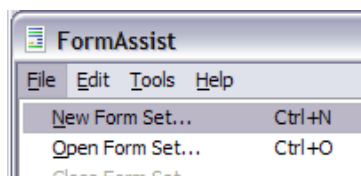


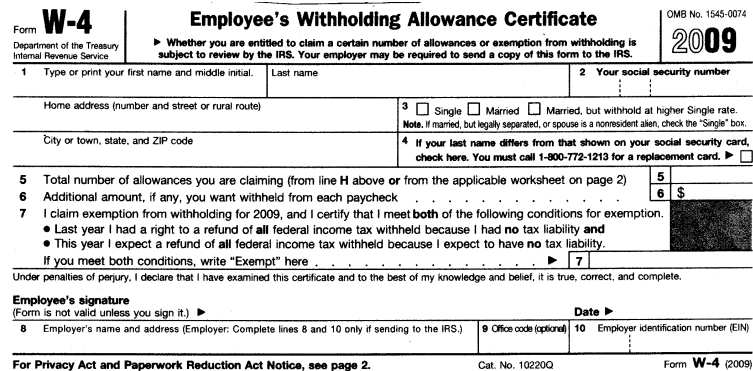
Figure 1: Creating a new Form Set

Throughout this article, we'll focus on the portions of the code that deal with forms processing, to the exclusion of extensive user interface code, exception handling, and other details that are part of the FormAssist sample. The important part of the code that creates the new Form Set (found within FormAssist) is shown below.

```
myFormSet = new FormDirector.FormSetFile(myFormDirector);
myFormSet.Name = "NewFormSet";
myFormSet.Filename = "NewFormSet.frs";
```

Note the use of FormDirector as the central repository and controller for all form and, as we'll see shortly, field information that will be needed during processing. FormDirector uses a straightforward XML format, which is fully documented in the SDK, to record the many settings that describe the entire forms processing procedure. This information is used during processing to control all the functions and settings of all of the components employed.

FormAssist will, of course, let you rename the Form Set, but we'll leave it as the default, since we only need one form set to demonstrate. For this example, I scanned a simple blank W4 form into a TIFF file.



Form W-4 **Employee's Withholding Allowance Certificate** OMB No. 1545-0074
 Department of the Treasury Internal Revenue Service **2009**

▶ **Whether you are entitled to claim a certain number of allowances or exemption from withholding is subject to review by the IRS. Your employer may be required to send a copy of this form to the IRS.**

1 Type or print your first name and middle initial. Last name 2 Your social security number

Home address (number and street or rural route) 3 Single Married Married, but withhold at higher Single rate. Note, if married, but legally separated, or spouse is a nonresident alien, check the "Single" box.

City or town, state, and ZIP code 4 If your last name differs from that shown on your social security card, check here. You must call 1-800-772-1213 for a replacement card. ▶

5 Total number of allowances you are claiming (from line H above or from the applicable worksheet on page 2) 5

6 Additional amount, if any, you want withheld from each paycheck 6 \$

7 I claim exemption from withholding for 2009, and I certify that I meet both of the following conditions for exemption:
 • Last year I had a right to a refund of all federal income tax withheld because I had no tax liability and
 • This year I expect a refund of all federal income tax withheld because I expect to have no tax liability.
 If you meet both conditions, write "Exempt" here ▶ 7

Under penalties of perjury, I declare that I have examined this certificate and to the best of my knowledge and belief, it is true, correct, and complete.

Employee's signature (Form is not valid unless you sign it.) ▶ Date ▶

8 Employer's name and address (Employer: Complete lines 8 and 10 only if sending to the IRS.) 9 Office code (optional) 10 Employer identification number (EIN)

For Privacy Act and Paperwork Reduction Act Notice, see page 2. Cat. No. 10220Q Form W-4 (2009)

Figure 2: The W-4 Template Image

Unless you've been working for the same company for 20 years, you probably recall filling out at least one of these. Generally, you'll want to save forms in TIFF format, as a bitonal Group 4 image, because it delivers lossless compression in a very small file size. The following code provides an overview of the process of adding the form into the form set.

Listing 1: Adding a Form to a Form Set

```
//always Deskew the template image prior to storage
ScanFix.Enhancements enhancements;
enhancements = new ScanFix.Enhancements();
enhancements.Options.Add(new ScanFix.DeskewOptions());
myScanFix.FromHdib(imageXView1.Image.ToHdib(false));
myScanFix.ExecuteEnhancements(enhancements);

//add the FileName as new Form in the Form Set
string newfileName = nodeName + ".frd";
FormDirector.FormDefinitionFile).Filename.ToUpper() ==
newfileName.ToUpper())

FormDirector.FormDefinitionFile myFormDef = new
    FormDirector.FormDefinitionFile(myFormSet);
FormDirector.TemplateImage myTemplateImage = new
    FormDirector.TemplateImage(myFormDirector);

// We'll store the original Image filename in OtherDataItems,
// so we can show it later as a property
FormDirector.DataItem myDataItem = new FormDirector.DataItem();
myDataItem.Type = PicConst.OriginalImage;
myDataItem.Content = formName;
myFormDef.OtherDataItems.Clear();
myFormDef.OtherDataItems.Add(myDataItem);

//take name from node in the tree
myFormDef.Name = nodeName;
myFormDef.Filename = newfileName;

// get the image from the ImagXpress view and save
myTemplateImage.Hdib = imageXView1.Image.ToHdib(false);
myFormDef.TemplateImages.Add(myTemplateImage);
myFormSet.FormDefinitions.Add(myFormDef);

dirtyFormSet = true; // remember this Form Set is not empty
```

Even though we require only one form in our example Form Set, that template will still be used to align the incoming images, drop out the parts of the image that are part of the blank form, and identify the zones of variable data that will be sent for recognition. Of course, you could also have many different forms in your set, in which case the incoming images would be compared to every one and matched with the best-fitting template, or rejected.

Defining Recognition Zones

There are three types of zones that we can define for recognition: OCR for machine-printed text, ICR for hand-printed text, and OMR for mark recognition. A fourth zone type called “clip” is provided for passing a chosen portion of the image to any outside process. FormAssist includes a drag-and-drop user interface for drawing these zones on your template form, and for defining in detail how each zone will be processed. The processing can include ScanFix functions (such as deskew, despeckle, dilate, erode, and character smoothing), Dropout parameters (including the dropout method and misalignment settings), and details for how recognition is performed (character sets for ICR/OCR, mark recognition thresholds for OMR, and many others).

First, we'll select the OCR field button (on the left, containing ABC, below) and draw an OCR field around the year on the form “2009”. FormAssist uses a light blue background to quickly identify OCR fields on the form. And we'll just click and drag a rectangle that includes the full text with at least a few pixels around it to allow for registration variances.

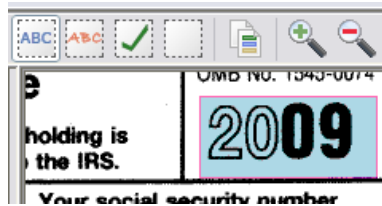


Figure 3: Drawing an OCR Recognition Zone

I named this field “Year”. If we select the ScanFix tab at the bottom, we can see a wide variety of available image cleanup functions that can be applied to this field. Each function, such as the currently-highlighted Deskew function, has various parameters that can be set individually for this field. I’ve chosen deskew and despeckle, moved them into that order using the up/down arrows, and left the default settings for both.

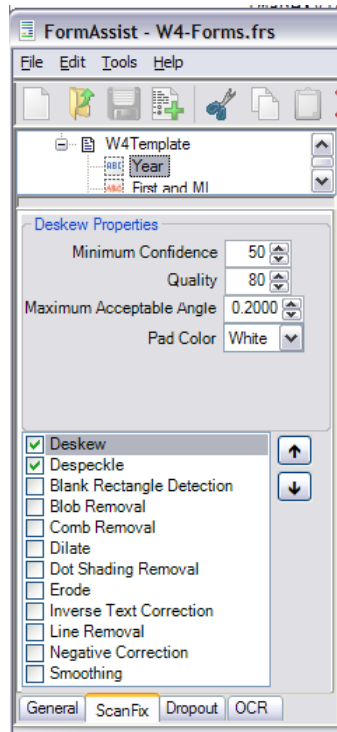


Figure 4: Applying ScanFix Cleanup to a Field

Now clicking the Dropout tab at the bottom, I am setting this to Clip rather than Dropout. This will leave this area in the incoming forms instead of dropping it out, so that I can OCR it and determine which year this W-4 form represents.

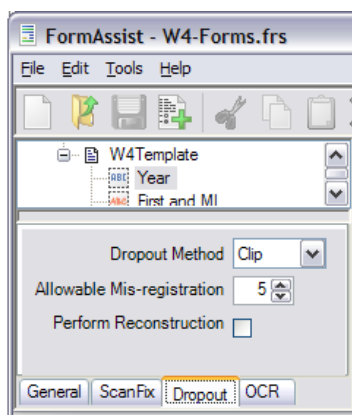


Figure 5: Setting Dropout method to Clip a Field

Next, I can click the OCR tab at the bottom and set up any specific parameters I might need to optimize the recognition in that zone.

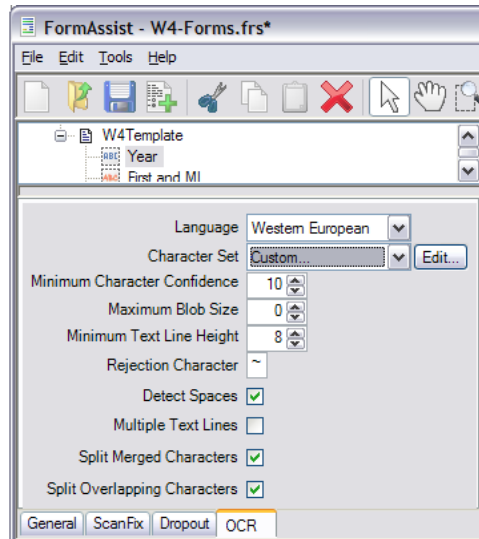


Figure 6: Setting OCR Parameters for a Field

In this case, can limit my recognition character set to only those three characters I'm expecting. Let's say I know that I only want to process W-4 forms from 2007, 2008 and 2009. I can limit recognition to only the numbers 0, 2, 7, 8, and 9. This will give me the highest possible confidence that anything found in that zone is actually one of these characters. I can create a separate custom character set for every field, if I wish.

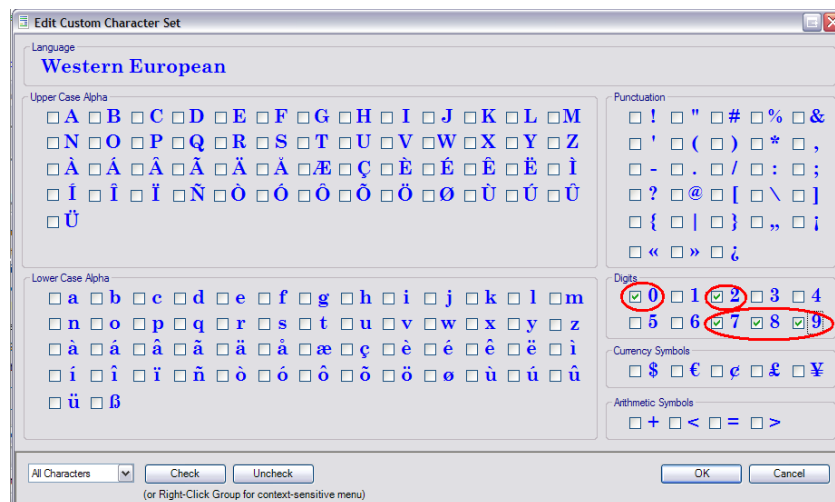


Figure 7: Selecting a Customer Character Set for a Field

Once we've selected the various types of processing to perform on our first OCR zone, it can be stored into FormDirector.

Listing 2: Adding a OCR Field to a Form

```
//Add a Form Field
FormDirector.FormDefinition mycurrentFormDef;
mycurrentFormDef = myFormSet.FormDefinitions[idxFormDef];
FormDirector.Field myField = new FormDirector.Field();
using (FormFix.DropoutProcessor dropProc = new
    FormFix.DropoutProcessor(myFormFix))
{
    FormDirector.DataItem myDataItem = new FormDirector.DataItem();
    //field type in OtherDataItem
    myDataItem.Type = PicConst.Type;

    //add Field Type XML
    myDataItem.Content = PicConst.OcrFieldType;
    myField.OtherDataItems.Add(myDataItem);

    //create the Dropout XML
    dropProc.DropoutMethod = FormFix.DropoutMethod.Dropout;
    dropProc.AllowableMisRegistration =
        (int)numericUpDownOCRMisReg.Value;
    dropProc.PerformReconstruction = checkBoxOCRReconstruc.Checked;
    tmpDropString = dropProc.WriteToStream();

    //create & add the OCR XML
    mySmartZone.Reader.Classifier =
        SmartZone.Classifier.MachinePrint;
    mySmartZone.Reader.MinimumCharacterConfidence =
        (int)numericUpDownOCRConf.Value;

    //set the default rejection character
    string s = textBoxRejection.Text;
    if (s == "")
    {
        s = "~";
        textBoxRejection.Text = s;
    }
    mySmartZone.Reader.RejectionCharacter = s[0];

    //set all OCR recognition parameters from on-screen settings
    mySmartZone.Reader.Area = new Rectangle(0,0,0,0);
    mySmartZone.Reader.CharacterSet =
        SmartZone.CharacterSet.AllCharacters;
    mySmartZone.Reader.CharacterSet.Language =
        SmartZone.Language.WesternEuropean;
```




```
mySmartZone.Reader.Segmentation.DetectSpaces =
    checkBoxOCRSpaces.Checked;
mySmartZone.Reader.Segmentation.MaximumBlobSize =
    (int)numericUpDownOCRBlock.Value;
mySmartZone.Reader.Segmentation.MinimumTextLineHeight =
    (int)numericUpDownOCRLine.Value;
mySmartZone.Reader.Segmentation.MultipleTextLines =
    checkBoxOCRMultiple.Checked;
mySmartZone.Reader.Segmentation.SplitMergedChars =
    checkBoxOCRMerged.Checked;
mySmartZone.Reader.Segmentation.SplitOverlappingChars =
    checkBoxOCROverlap.Checked;

// add the OCR data item to FormDirector
tmpOCRString = mySmartZone.WriteToStream();
FormDirector.DataItem myOCRDataItem = new
FormDirector.DataItem();
myOCRDataItem.Type = PicConst.RecognitionOp;
myOCRDataItem.Content = tmpOCRString;
myField.Operations.Add(myOCRDataItem);

//add Dropout xml
myField.Construction.Type = PicConst.DropoutOp;
myField.Construction.Content = tmpDropString;
}

// add the name of the field (make it unique using a sequence
number)
myField.Name = "Field " + nodeFieldSequenceNumber.ToString();
fieldNode.Text = "Field " + nodeFieldSequenceNumber.ToString();

// actually add the field to the form definition file
mycurrentFormDef.Fields.Add(myField);
```

Now we're ready to add our first employee information gathering field, an ICR zone that will be used to collect the first name and middle initial of new employees who have submitted Withholding Allowance forms. FormAssist uses an orange background to quickly identify ICR fields on the form. We simply click the second ABC box and draw a rectangle around the area that includes that data.

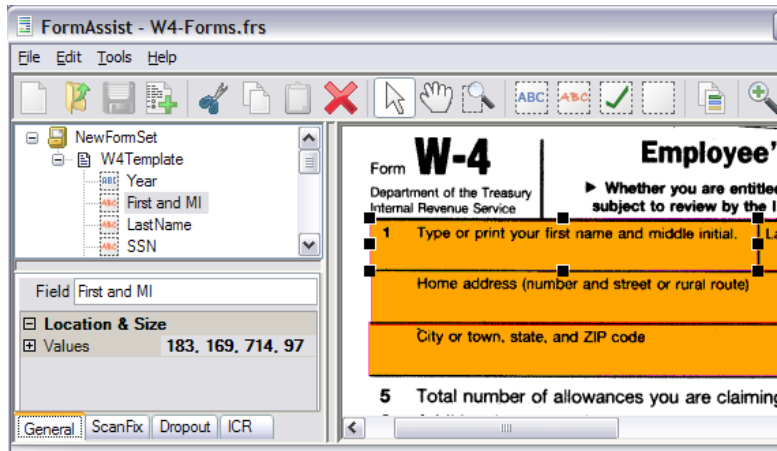


Figure 8: Adding an ICR Field to a Form

Notice that the guide text “Type or print your first name and middle initial.” is fully included in the zone! If this text were to be sent for ICR, it could cause lots of extraneous results. This is where the magic of form dropout, as selected on the Dropout tab below, comes into play. When chosen, anything that appears on the template form will be removed before the field is sent for recognition. The Perform Reconstruction parameter should always be set, to produce clean results after dropping out the template, leaving only the variable data (the info our new employee printed on his blank form) remaining from the input image.

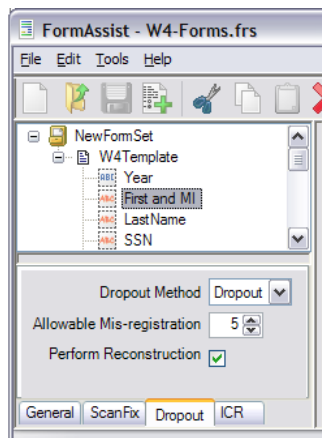


Figure 9: Setting Form Dropout for Field Recognition

Since first names don't normally include numbers, we can improve our results by restricting our ICR character set for this field to “All Alphas,” as shown below. The other common character sets are also shown in the pull-down list. As we showed previously, the Custom setting is used to select any desired combination of characters for recognition.

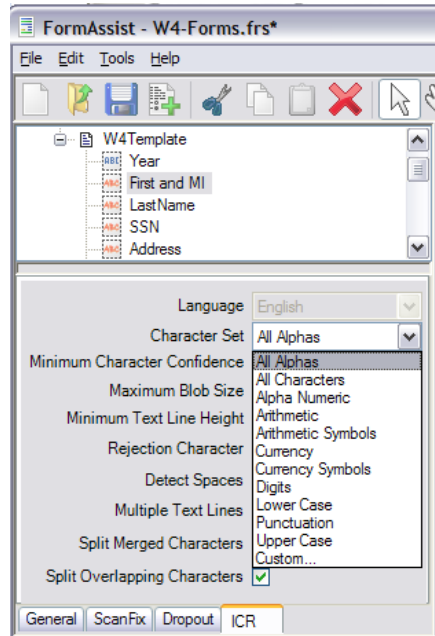


Figure 10: Setting an ICR zone for “Alpha Only” recognition

The rest of the ICR fields are created in the same fashion. The SSN field, for example, can only hold digits, and the Exempt field can only contain the letters E, X, M, P, and T. The actual code to add this ICR field is very similar to the code to add the OCR field above.

The last type of field we will enter is an OMR check-box. These are drawn and displayed with light green rectangles in FormAssist. Again, you simply select the OMR drawing tool (with the green check-mark), and draw the rectangle on the form.

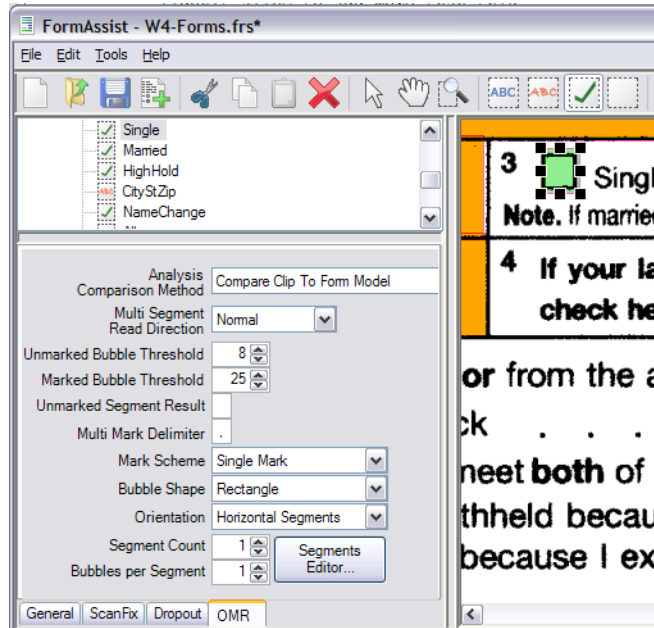


Figure 11: Setting Parameters for Mark (OMR) Recognition

We won't go over all of the details, but here you have a great variety of options on how marks can be defined for recognition, including mark thresholds and return values, various bubble shapes, and the ability to describe arrays (rows and columns) of marks as a single object. The OMR feature can also be used to detect the presence of a signature, as required on this form. A snapshot of the relevant code to add the OMR field follows.

Listing 3: Adding an OMR Field to a Form

```
//Add an OMR Field
resetProperties(propType.OMR);
myDataItem.Content = PicConst.OmrFieldType;
myField.OtherDataItems.Add(myDataItem);

//create the Dropout XML
dropProc.DropOutMethod = FormFix.DropOutMethod.Clip;
dropProc.AllowableMisRegistration =
(int)numericUpDownOMRMisReg.Value;
dropProc.PerformReconstruction = checkBoxOMRReconstruc.Checked;
tmpDropString = dropProc.WriteToStream();

//create & add the OMR XML
using (FormFix.OmrProcessor omrProc = new
FormFix.OmrProcessor(myFormFix))
{
    omrProc.Area = new Rectangle(0,0,0,0);
    omrProc.AnalysisComparisonMethod =
FormFix.OmrAnalysisComparisonMethod.None;
    omrProc.MultiSegmentReadDirection =
        FormFix.OmrMultiSegmentReadDirection.Normal;
    omrProc.MarkedBubbleThreshold =
(int)numericUpDownOMRMBubble.Value;
    omrProc.UnmarkedBubbleThreshold =
(int)numericUpDownOMRUBubble.Value;
    omrProc.UnmarkedSegmentResult = textBoxOMRUSegment.Text;
    omrProc.MarkScheme = FormFix.OmrMarkScheme.SingleMark;
    omrProc.MultiMarkDelimiter = textBoxOMRDelim.Text;
    omrProc.BubbleShape = FormFix.OmrBubbleShape.Circle;
    omrProc.Orientation =
FormFix.OmrOrientation.HorizontalSegments;
    tmpOMRString = omrProc.WriteToStream();
    FormDirector.DataItem myOMRDataItem = new
FormDirector.DataItem();
    myOMRDataItem.Type = PicConst.OmrOp;
    myOMRDataItem.Content = tmpOMRString;
    myField.Operations.Add(myOMRDataItem);
}
// add the name of the field (make it unique using a sequence
number)
myField.Name = "Field " + nodeFieldSequenceNumber.ToString();
fieldNode.Text = "Field " + nodeFieldSequenceNumber.ToString();

// actually add the field to the form definition file
mycurrentFormDef.Fields.Add(myField);
```



The rest of the fields on this form are essentially the same as the sample OCR, ICR and OMR fields we have just defined. Without showing the code to obtain the name for your Form Set file from a dialog box, the following is the heart of the code to save your completed forms (or just one form, in our case) to a Form Set file.

Listing 4: Saving the Form Set

```
//Save the current Form Set
int myFormCount = myFormSet.FormDefinitions.Count;
for (int i=0; i<myFormCount; i++)
{
    FormDirector.FormDefinitionFile myFormDef =
        (myFormSet.FormDefinitions[i] as
        FormDirector.FormDefinitionFile);
}
```

Processing Live Forms

Although the processing of actual forms is often done with a separate application, the FormAssist demo lets you manually try out some of your live images, so that you can confirm that everything's set up correctly and you can fine-tune parameters to optimize your results. This processing code within the demo can be used to create your own production system, which might also include subsystems for document scanning, or for manual review of low-confidence data fields by operators. Since the exact location of each suspect field is known, it's quite easy to build a Key From Image (KFI) user interface that zooms in on the correct part of the image, and allows the operator to type any corrections before the data is stored in a permanent database.

FormAssist includes a lot of reusable code that you can use to build a production-level processor to cycle each input image through the form matching, dropout, cleanup and recognition actions stored by FormDirector as XML streams. The following code snippets are intended to demonstrate how many of the individual operations might be implemented within your code, rather than showing the full solution.

Listing 5: Processing Forms – Image Cleanup

```
// Method to process a filled image, first performing image
identification and
// alignment, then processing each field on the form if a match was
found
public void ProcessImage(System.Drawing.Image inputImage)

// Set the image into a form image
filledImage =
FormFix.FormImage.FromBitmap((System.Drawing.Bitmap)inputImage, FF);
filledImage.HorizontalResolution = inputImage.HorizontalResolution;
filledImage.VerticalResolution = inputImage.VerticalResolution;
imageToProcess = filledImage;

// Do the full-page ScanFix enhancements (rather than field cleanup)
ScanFix.Enhancements enhancements;
enhancements = new ScanFix.Enhancements();

// get the XML list of cleanups from FormDirector
enhancements.ReadFromStream(enhancementXML, SF);

// Set the image to be enhanced into ScanFix, then enhance
SF.FromHdib(frmImage.ToHdib(false));
SF.ExecuteEnhancements(enhancements);

// Set the results back into the form image
return FormFix.FormImage.FromHdib(SF.ToHdib(true), true, FF);
```

Listing 6: Processing Forms – Form Identification, and Alignment

```
// Set up the form identification processor
idProcessor = new FormFix.IdentificationProcessor(FF);

// set up the event handlers for processing the form
idProcessor.ReadChecksum += new
FormFix.ReadChecksumEventHandler(formIdReadChecksum);
idProcessor.ReadDataItem += new
FormFix.DataItemEventHandler(formIdReadDataItem);
idProcessor.WriteDataItem += new
FormFix.DataItemEventHandler(formIdWriteDataItem);

// get the FormSet setting for FormFix for identifying a form
idProcessor.ReadFromStream(FS.Identification.Content);
idProcessor.MaximumIdentificationBestMatches = maximumBestMatches;

// Add the form models to the identifier
```



```
for (int i = 0; i < FS.FormDefinitions.Count; i++)
{
    formModel = new FormFix.FormModel(FF);
    formModel.Tag = FS.FormDefinitions[i];
    formModel.Name = FS.FormDefinitions[i].Name;

    formModel.ReadDataItem += new
        FormFix.DataItemEventHandler(formModelReadDataItem);
    formModel.ReadFormImage += new
        FormFix.ReadFormImageEventHandler(formModelReadFormImage);
    formModel.ReadChecksum += new
        FormFix.ReadChecksumEventHandler(formModelReadChecksum);
    formModel.WriteDataItem += new
        FormFix.DataItemEventHandler(formModelWriteDataItem);

    idProcessor.FormModels.Add(formModel);
}
// Identify the unknown image
idResult = idProcessor.Identify(unknownImage);

if (idResult.State != FormFix.IdentificationState.NoMatchFound)
{
    // Match found, align the image
    alignedImage = idResult.RegistrationResult.AlignImage(unknownImage);

    // After alignment, process each field
    // All fields are described in the formDef
    formDef = (FormDirector.FormDefinitionFile) idResult.FormModel.Tag;
    for (int i = 0; i < formDef.Fields.Count; i++)
    { //processes each field according to formDef instructions
        ProcessField(formDef.Fields[i], imageToProcess);
    }
}
```

Listing 7: Processing Forms – Field Dropout

```
//for each field, Drop out the form if requested
if (field.Construction.Type == PicConst.DropoutOp)

//create a dropout processor to remove the form from the background
dropOutProcessor = new FormFix.DropOutProcessor(FF);
dropOutProcessor.ReadFromStream(field.Construction.Content);
dropOutProcessor.Area = field.Location;

// Dropout processor is executed against the enhanced or original
filled image
if (enhancedImage != null)
    dropOutResult = dropOutProcessor.CreateImageOfField(enhancedImage,
        idResult.RegistrationResult);
```



```
else  
    dropOutResult = dropOutProcessor.CreateImageOfField(filledImage,  
        idResult.RegistrationResult);
```

Listing 8: Processing Forms – Field Cleanup and Recognition

```
// Next clean up the individual field  
ScanFix.Enhancements enhancements;  
enhancements = new ScanFix.Enhancements();  
    enhancements.ReadFromStream(field.Operations[idxOp].Content, SF);  
  
SF.FromHdib(imageToProcess.ToHdib(false));  
SF.ExecuteEnhancements(enhancements);  
imageToProcess = FormFix.FormImage.FromHdib(SF.ToHdib(true), true, FF);  
thisFieldResult.EnhancedImage = imageToProcess;  
  
// If OMR field, do mark recognition  
idxOp = field.Operations.GetIndexOfType(PicConst.OmrOp);  
if (idxOp >= 0)  
    if (omrProcessor == null)  
        omrProcessor = new FormFix.OmrProcessor(FF);  
  
// get the OMR settings from the form definition file for this field  
omrProcessor.ReadFromStream(field.Operations[idxOp].Content);  
  
// Depending on the OMR settings from choose the type of comparison  
switch (omrProcessor.AnalysisComparisonMethod)  
{  
    case FormFix.OmrAnalysisComparisonMethod.None:  
        // Use the image clip  
        omrProcessor.Area = new System.Drawing.Rectangle(0,0,0,0);  
        imageToOmr = imageToProcess;  
        break;  
  
    case FormFix.OmrAnalysisComparisonMethod.CompareClipToFormModel:  
        // Use the image clip  
        omrProcessor.OriginOfClip = new System.Drawing.Point  
            (field.Location.X, field.Location.Y);  
        omrProcessor.Area = new System.Drawing.Rectangle(0,0,0,0);  
        imageToOmr = imageToProcess;  
        omrProcessor.FormModel = idResult.FormModel;  
        break;  
  
    case  
        FormFix.OmrAnalysisComparisonMethod.CompareFullImageToFormModel:  
        // Use the aligned image, since we are comparing to the form image  
        omrProcessor.Area = field.Location;
```

```

imageToOmr = alignedImage;
omrProcessor.FormModel = idResult.FormModel;
break;
}

// Perform the OMR
thisFieldResult.OmrResult = omrProcessor.AnalyzeField(imageToOmr);

// Perform OCR for the Field (similar for ICR)
idxOp = field.Operations.GetIndexOfType(PicConst.RecognitionOp);
if (idxOp >= 0)
{
    SZ.ReadFromStream(field.Operations[idxOp].Content);
    thisFieldResult.OcrResult =
        SZ.Reader.AnalyzeField(imageToProcess.ToHdib(false));
}

```

If you click on the Aligned Image thumbnail, you can see exactly how dropout performed. Only the items shown in red still remain after the template form was dropped from the scanned image. This contains the variable data that is recognized to eventually populate your database.

Form W-4		Employee's Withholding Allowance Certificate		OMB No. 1545-0074
Department of the Treasury Internal Revenue Service		<p>▶ Whether you are entitled to claim a certain number of allowances or exemption from withholding is subject to review by the IRS. Your employer may be required to send a copy of this form to the IRS.</p>		2009
1	Type or print your first name and middle initial.	Last name	2	Your social security number
	ROBERT J.	SANDERS		317-92-0437
Home address (number and street or rural route)		3	<input type="checkbox"/> Single <input checked="" type="checkbox"/> Married <input type="checkbox"/> Married, but withhold at higher Single rate. <small>Note. If married, but legally separated, or spouse is a nonresident alien, check the "Single" box.</small>	
2197 EDMONT BLVD.			4	
City or town, state, and ZIP code		If your last name differs from that shown on your social security card, check here. You must call 1-800-772-1213 for a replacement card. ▶ <input type="checkbox"/>		
SPRINGFIELD, MT				
5	Total number of allowances you are claiming (from line H above or from the applicable worksheet on page 2)	5	4	
6	Additional amount, if any, you want withheld from each paycheck	6	\$ 250	
7	I claim exemption from withholding for 2009, and I certify that I meet both of the following conditions for exemption. <ul style="list-style-type: none"> • Last year I had a right to a refund of all federal income tax withheld because I had no tax liability and • This year I expect a refund of all federal income tax withheld because I expect to have no tax liability. If you meet both conditions, write "Exempt" here			
		7	EXEMPT	
Under penalties of perjury, I declare that I have examined this certificate and to the best of my knowledge and belief, it is true, correct, and complete.				
Employee's signature <small>(Form is not valid unless you sign it.) ▶</small>		Date ▶		
Robert J. Sanders		21 JUN 09		
8	Employer's name and address (Employer: Complete lines 8 and 10 only if sending to the IRS.)	9	Office code (optional)	10
		Employer identification number (EIN)		
For Privacy Act and Paperwork Reduction Act Notice, see page 2.				
Cat. No. 10220Q			Form W-4 (2009)	

Figure 12: Aligned Image, Showing Variable Data in Red

If you click on the Field Results tab, you can see the actual cleaned image clips for each zone. Note how the guide text (Your social security number) has been completely removed, resulting in accurate recognition:

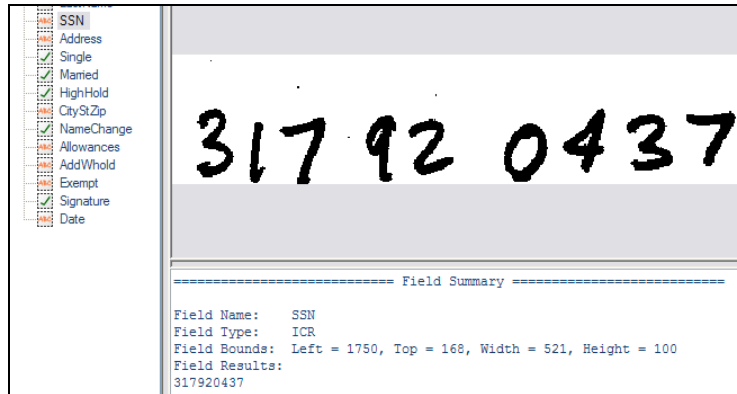


Figure 13: Field Results, Showing Clipped and Cleaned Zone

Conclusion

The sample code in FormAssist serves two important purposes for any developer wishing to build an application to extract data from structured forms. First, it can be used to create form templates defining recognition zones for each type of form you'll use in your document workflow. Second, it serves as the starting point for your own form definition application and your custom batch form processing solution. As previously mentioned, the code shown here represents some of the key steps involved. The entire sample, and all referenced components, can be downloaded here: [Download Accusoft Pegasus FormSuite SDK](#)

You can find Accusoft Pegasus product downloads and features at www.accusoft.com. Please contact us at sales@accusoft.com or support@accusoft.com for more information.

About Accusoft Pegasus

Founded in 1991 under the corporate name Pegasus Imaging, and headquartered in Tampa, Florida, Accusoft Pegasus is the largest source for imaging software development kits (SDKs) and image viewers. Imaging technology solutions include barcode, compression, DICOM, editing, forms processing, OCR, PDF, scanning, video, and viewing. Technology is delivered for Microsoft .NET, ActiveX, Silverlight, AJAX, ASP.NET, Windows Workflow, and Java environments. Multiple 32-bit and 64-bit platforms are supported, including Windows, Windows Mobile, Linux, Sun Solaris, Mac OSX, and IBM AIX. Visit <http://www.accusoft.com> for more information.



About the Author

Paul B. Firth, Product Manager

Paul has been helping to guide the overall product strategy for Accusoft Pegasus since 2005. In this role he works closely with the Sales and R&D teams to identify and satisfy product needs in the highly-dynamic software tools market. Prior to joining Accusoft Pegasus, he has led high-tech marketing teams and managed teams of both software and hardware developers. Paul holds a Masters in Business Administration from New York University, and Bachelor of Science degree in Electrical Engineering from the University of Rochester.