Adobe Central to AEM Forms Migration Guide

A technical guide for migrating to AEM forms

Legal notices

Table of Contents

1. Introduction

This document is intended for technologists who have experience building document generation and print solutions with the Adobe Central Output Server family of products (Central), including Adobe Central Pro Output Server, Web Output Pak, and Output Designer. It will help individuals whose organizations are either considering, or beginning a migration to Adobe AEM forms by comparing and contrasting the capabilities of the two product lines, describing varying migration scenarios and available tools, identifying Central features that do not exist in AEM forms , and describing some of the challenges and remedies.

It is assumed that readers have prior hands-on experience with the Central family of products and are proficient in their knowledge and understanding of these products. In particular, it is assumed that readers have an understanding of how forms (also known as templates) are designed using Adobe Output Designer, how dynamic forms are constructed using subforms, what the role of preambles is, and how the field-nominated data format utilized by Central is used to drive document generation and printing.

In terms of knowledge required to understand AEM forms, it is assumed that readers have some familiarity with XML and related standards and tools, an awareness of the major features of AEM forms, and some experience with AEM forms Designer.

Chapter 2, *Product and Technology Overview*, provides an overview intended to help readers quickly understand basic differences between Central and AEM forms Output Service; the remaining chapters provide significantly greater detail.

Goals and Scope

This document details how the features of the Adobe Central Output Server family of products compare to AEM forms, often in a technical way. Assuming that readers have previous experience developing document generation and printing solutions with Central, this document seeks to leverage their existing experience and knowledge so they can quickly understand how equivalent solutions may be developed with AEM forms.

Central and AEM forms Output Service are intended to address the same core document generation and printing goals, but they differ in terms of their technological foundations, breadth of functionality, use of standards, and countless details. Nonetheless, the common heritage and goals of these products provides for a common basis of understanding and learning.

This document is not a tutorial on the overall AEM forms product or any of its solution components, nor can it replace any of the product documentation or samples provided with AEM forms. Instead, this document provides an introduction to many aspects of AEM forms, and then refers readers to the AEM forms Output Service product documentation for additional learning.

The document does not attempt to discuss Output Pak for SAP; AEM forms does not provide SAP integration. These customers are encouraged to investigate “SAP Interactive Forms by Adobe”; information available here http://www.adobe.com/ap/enterprise/partners/sap.html

Organization of this Document

This document is divided into the following sections:

- Chapter 2, *Product and Technology Overview*, provides a high-level overview of the major aspects and solution components of the Central family of products and AEM forms. This chapter offers a quick understanding of the basic differences between Central and AEM forms Output Service

- Chapter 3, *Central Migration Bridge*, provides a detailed examination of the Central Migration Bridge capability of AEM forms. The Bridge provides a way for existing Output Designer forms, transformation definitions and field-nominated data to be used in the AEM environment. Use of the Bridge provides a quick transition to the AEM p l a t f o r m for your Central applications; over time applications can be migrated in a staged fashion matching business needs and demands; new applications can be developed and run on the same AEM server taking advantage of Lifecycle's many capabilities.

- Chapter 4*, Forms*, provides a detailed examination of forms in Central and AEM forms Output Service from two perspectives. First, AEM forms Designer (the form designer within AEM forms) has the capability to import Central forms created with Output Designer. How this import capability works, the results it produces, and some of the areas that may require additional attention once the import is complete are discussed. Second, this chapter offers an overview of the major features of forms created with AEM forms Designer and how those capabilities are similar to or different from forms created with Output Designer.

- Chapter 5, *Data*, provides information relevant to transitioning from the primary data format of Central (the field-nominated format) to the XML data capabilities available with forms created with AEM forms Designer. First, it discusses XML as a data format, along with comparisons to the field-nominated format in Central. Then, it explains how AEM Form® Output Service combines XML data with forms to generate output.

- Chapter 6, *Document Generation*, explores important aspects of how AEM forms Output Service and AEM forms Designer can be used to generate documents and printed output. Topics of discussion include common methods of invoking AEM Forms ® Output Service, how to test and preview forms in AEM forms Designer, how formats and print devices are handled and configured, and specific aspects of output generation such as font and paper handling.

- Chapter 7, *Web Output Pak*, offers a brief discussion of how Web Output Pak relates to AEM forms Output Service.

- Chapter 8, *Hosting Environment*, provides a brief discussion of the different platform and hosting requirements of Central and AEM forms.

- Chapter 9, *Field-Nominated Commands*, lists most of the commands that compose the field-nominated format in Central and directs readers to relevant resources for more information on how to accomplish similar tasks, such as the original field-nominated command.

2. Product and Technology Overview

Why Move to AEM Forms?

It is important to note that while this document seeks to explain the effort to migrate applications from the Central Output Server family to AEM forms, Adobe acknowledges that migration represents work which requires planning and is often carried out over time as part of an overall application support lifecycle. It is with this understanding that Adobe is committed to the continued sale and support of the Central Output Server family so that clients have the flexibility they need to either maintain their existing installations or migrate to the AEM platform on a schedule that meets their organizational requirements.

Current information regarding the Central family as well as support and migration options can be found on Adobe's website.

- www.adobe.com/products/server/outputserver
- www.adobe.com/support/programs/policies/policy_enterprise_lifecycle.html

Adobe Experience Manager (AEM) provides an easy-to-use solution to create, manage, publish, and update complex digital forms while integrating with back-end processes, business rules, and data.

AEM forms combine form authoring, management, and publishing along with correspondence capabilities, document security, and integrated analytics to create engaging end-to-end experiences. Designed to work across web and mobile channels, AEM forms can be efficiently integrated into your business processes, reducing paper processes and errors while improving efficiency.

AEM forms leverages and extends the capabilities of your existing investments in XFA forms and Adobe LiveCycle solution. For details, see AEM forms - securing and extending your existing forms ecosystem. In large enterprises, forms are often created once and reused by copying to a content management system. Keeping a large database of forms up-to-date and making forms discoverable can be a considerable challenge. AEM provides a customizable Forms Portal that ensures that customers find and access the forms they need across both web and mobile channels.
AEM forms provides forms management tools that not only lets you manage adaptive form, but XFA forms, PDF forms, and related assets as well. For more information, see Introduction to managing forms.

AEM Forms includes the following solution components:

Interactive Forms:

- Forms Service – Create and deploy interactive XML-based forms as HTML, PDF.
- AEM Reader Extensions Doc service – Fill in, sign, comment on, or save Adobe PDF files using only Adobe Reader software.

Document Security:

- AEM forms DocAssurance Service – Automate the validation of digital signatures in PDF documents.

Document Generation:

- AEM forms Output Service–Dynamically generates personalized documents for processes that require on-demand or medium volume batch output support.
- AEM forms Assembler Service - Assemble PDF packages from existing files or pages.

Key capabilities

To sum up, AEM forms provide powerful form management features, like the following, that reduce manual processes and increase customer satisfaction.

- A centralized Forms Portal for designing and deploying dynamic forms, including PDF, HTML5, and adaptive forms
- An easy-to-use graphical user interface to let business users easily import, manage, preview, and publish forms
- A responsive forms directory with powerful search features using keywords, tags, and metadata
- Dynamic detection of a user's device and location to render the form appropriately across web and mobile channels
- Integration with Adobe Analytics to effectively measure form usage metrics
- Integration with Adobe EchoSign integration or Scribble to electronically encrypt documents containing confidential information
- Automated form-publishing capabilities and the ability to deliver timely, personalized, and consistent communication through multiple channels

Form Design

Central forms are designed using Output Designer, which offers a graphical environment for designing both the visual aspects of a form, and the behavioral characteristics of the form. Forms created with Output Designer are saved in an **.ifd** binary format, and deployed to Central in an **.mdf** binary format.

Forms are also designed using the graphically rich environment provided by AEM forms Designer. The set of tools and capabilities offered within AEM forms Designer are significantly greater than Output Designer. In many cases, features and capabilities that required hand-coding of preambles or data in Central are now available as equivalent first-class features of AEM forms Designer. Forms created with AEM forms Designer are saved in an **.xdp** format, which is an XML markup format, when they are to be deployed to AEM.

Chapter 4, *Forms*, provides a more detailed discussion of topics related to migrating and designing forms.

Data Integration

While Central has supported XML data as an input format for many years, the primary data format Central supports is known as the **field-nominated format**. Central also provides support for a number of legacy data formats and includes a software for assisting with transforming other non-supported legacy formats to the field-nominated and XML format: the Visual Transformation Editor and Transformation Agent which generate and execute transformation definition files; **.tdf** format

Forms created with AEM forms Designer can connect with data sources such as databases and web services. However, the primary supported data format is generic XML data that may optionally conform to a custom schema of your choosing. Aside from this emphasis on XML, there is no specific XFA data format.

AEM forms Designer provides a robust set of tools for integrating XML data with forms. There is no equivalent to the Visual Transformation Editor in the AEM forms suite of solution components.

Chapter 5, *Data*, provides a more detailed discussion of topics related to moving from field-nominated data, to XML data.

Processing

Central is a processing engine that receives data, optionally combines it with a form, and invokes one or more software components (known as agents) according to process descriptions contained within a text file known as the Job Management Database (JMD).

Central typically integrates with other software systems by monitoring file-system directories (known as collector directories). Central can also leverage agents as adapters between its collector directories and other means of communication, such as printer queues or e-mail.

Unlike Central, which is a native application designed to run on several platforms .AEM forms is an AEM application deployed as a package. The package contains services or API providers deployed in AEM OSGi container and servlets or JSPs. These servlets provide front-end and REST API functionalities that are managed by AEM Sling framework. AEM forms OutputCentral Service is used to invoke Central and is publicly available for consumption by custom code co-deployed in AEM. These distinguishing architectural features of AEM forms provide a basis for significant scalability and integration opportunities. However, this approach also firmly defines AEM forms as a server application not intended to be deployed to desktop systems, except for development and testing. For more information on this topic, see Chap-ter 7, Hosting Environment.

AEM Forms can be used with Core AEM services like AEM Workflows which can be used to automate Experience Manager activities. Workflows consist of a series of steps like manipulating XML data, generating documents, sending output to printers, and more. Custom activities can be developed and incorporated into workflows.

Central has very different performance characteristics compared to AEM forms Output Service. As a native application with limited layout capabilities it will format individual and batches of documents significantly faster than AEM forms, anecdotally at least 2X faster depending on scenario; if FNF/DAT data is in use rather than XML date up to 4X faster and if there is no need for “Page y of n” page num-ber formatting which requires 2 formatting passes for Central up to 8X faster. These are very rough rules of thumb and real results will certainly vary – it is highly recommended that performance testing / sizing be part of any migration effort especially where high volumes 100,000+ pages/day are a factor. AEM forms however provides far more robust access to computing resources, no special set up is required to access the power of available CPU/Cores/Servers as required for Central where multiple instances must be established and work flow managed to balance resource use across the computing platform.

Chapter 6, *Document Generation*, provides a more detailed discussion of how generating documents with AEM forms differs from Central.

Comparison Summary

The following table provides an additional comparison of Adobe Central Pro Output Server v5.7 and AEM forms Output Service. Where noted, it compares capabilities delivered through related products such as Web Output Pak or AEM Form® Forms Service.

Feature	**Central Pro Output Server 5.7**	**AEM Forms**
Hosting environment	Native application with limited opportunities for scaling and load balancing	It is an AEM application deployed as a package. The package contains services or API providers deployed in AEM OSGi container and servlets or JSPs. These servlets provide front-end and REST API functionalities that are managed by AEM Sling framework.
Operating system	Windows 2000/2003/2008 Linux Red Hat / SUSE Solaris AIX HP-UX OS/400 (last version 5.5)	Windows 2003/2008 - JBOSS, WebSphere or Weblogic Linux Red Hat / SUSE - JBOSS, WebSphere or Weblogic Solaris - WebSphere or Weblogic AIX - WebSphere Not supported Not supported
Invocation	Command-line, collector directory, e-mail	• Through wrapper OSGI service • ECMA script used within AEM Workflows • By Implementing workflowProcess interface and using workflows • Through Servlets,JSP Or REST calls
Design environment	Output Designer	AEM forms Designer Rich WYSIWYG modern GUI, drag-and-drop XML schema integration, declarative sub-form design, scripting models for calculations and logic, widow and orphan control, page n of m, nested tables, spell checking, and more
Form management	Object library PDF preview, test print	Integrated repository, form and fragment library (fragments are referenced form objects) PDF preview, test print (Print Form With Data)
Data handling	XML, field-nominated data, comma-delimited, fixed-record Other formats accommodated with Visual Transformation Editor and Transformation Agent.	XML data, transformation between XML formats provided with support for XSLT Not supported

Feature or Capability	**Central Pro Output Server 5.7**	**AEM forms Output Service**
Output formats	PDF PostScript PCL5e, PCL5c, PCL6(XL) Zebra Label – ZPL Intemec Label – IPL TEC Label – TPCL Datamax Label – DPL Monarch Label Windows driver Fax drivers HTML (Web Output Pak only)	PDF, PDF/A-1a, PDF/A-1b PostScript PCL 5e, PCL5c Zebra Label – ZPL Intemec Label – IPL TEC Label – TPCL Datamax Label – DPL Not supported Not supported HTML5 (AEM Mobile & Adaptive Forms)
Ability to assemble individual forms into documents	Use ^form field-nominated command to use individual forms when formatting one document	Assembler service • Insert one or more subforms at insertion points defined in forms by AEM forms Designer; generating a customized .xdp for formatting with AEM forms Output Service • Combine individual forms (including those with insertion points) into one form (.xdp) for formatting with AEM forms Output Service.
Process Design Environment	Central Pro Output Server Job Management Database (JMD), Web Output Pak, XML Processing Rules (XPR)	AEM workflows using the AEM workflow UI
PDF conversion formats; Convert PDF to	None	• PostScript for server-based PDF printing to non-PDF printers • PDF/A and Single & multipage to support archiving requirements
Ability to assemble multiple PDF documents	None	Assembler service (part of AEM forms) • Combine PDF files or pages • Combine multiple documents into one single file, or create a document portfolio with multiple documents – add a flash based navigator for portfolio contents. • Automatically generate a hyperlinked table of contents • Add watermarks, backgrounds, overlays • Add headers, footers, or numbering • Add, remove, rotate, and scale pages • Import, export, and manipulate attachments, annotations, links, and bookmarks in XML • Manipulate document metadata

Related Documentation

Two major categories of additional documentation referenced in this document provide deeper levels of understanding: documentation related to the AEM forms product and documentation related to the various standards leveraged by AEM forms Output Service.

Additional product-related documentation includes:

- *XML Forms Architecture Specification*

  The technology underlying AEM forms Designer, the forms it produces, and the manner in which these forms behave is largely a result of a family of XML markup languages known as the *XML Forms Architecture* (XFA). This markup language is defined by a formal specification document available from the Adobe website at: http://partners.adobe.com/public/developer/xml/index_arch.html

Additional standards-related documentation includes:

- *PDF Reference, Sixth Edition, version 1.7*

  A formal specification of PDF is available from the Adobe website at: http://www.adobe.com/devnet/pdf/

- *Internationalization*

  The International Organization for Standardization (ISO) provides several standards associated with the accurate interchange of locale-specific information, such as identifying locales and languages (ISO 639-1, ISO 3166-1) and expressing currencies (ISO 4217) and dates and time (ISO 8601). These standards are used by XML, XFA, and solutions components of AEM forms Output Service.

- *Unicode*

  Unicode is a standard for the encoding and interchange of characters and symbols used in the languages of the world. XML leverages Unicode for all content. More information on the Unicode standard can be found on the Unicode website at: www.unicode.org

- *XML 1.1*

  AEM forms Output Service utilizes XML and XML-related technologies throughout. The formal specification of XML is available from the World Wide Web Consortium (W3C) website at: www.w3.org/TR/xml11

- *XSL Transformations (XSLT), version 2.0*

  XSLT is a markup language and a technology used to transform XML. Within the context of AEM forms Output Service, XSLT is often referenced as a mechanism for transforming XML data into a format more suitable for use with a particular XFA template.

3. Central Migration Bridge

This chapter explores the Central Migration Bridge capability of the AEM forms OutputCentral service. The Central Migration Bridge provides a way to continue to use the input data files that you have been using with Central, while bringing your forms and other processing into AEM. It implies that you will need to make minimal to no changes to your back-end systems to gain the benefits of AEM.

Overview of the Central Migration Bridge Services

Central Migration Bridge includes:

- **centralTransformation** service that runs the Central Transformation Agent (jftrans.exe).
- **centralDataAccess** service that will read a .DAT file and liberate important values from it into an XML structure that can then be parsed via an XPath expression. This will enable the user to process some of the options that are passed to the .DAT file. It is **not** intended to be a general purpose .DAT to XML converter. Instead it is intended to allow someone to get access to data in the incoming data stream that may be used afterwards in the AEM Workflow that invoked it.

Central Migration Bridge requires Central Pro 5.7. If Central Pro 5.7 is installed in the default location, then the Central Migration Bridge will locate it automatically. If Central Pro 5.7 is installed into some location other than the default location, then the alternate location must be configured during installation or via the AEM Web Console (Felix Configuration Manager). The Central Migration Bridge does not need Central to be running in order to function; it just needs access to the Central Agents executables utilized by the Central Migration Bridge services and to the INI files corresponding to those agents. For simplicity of configuration, it is recommended that Central Pro 5.7 be installed in the default installation location before AEM is installed and configured. Please refer to the Central installation documentation for installation instructions.

How the Central Migration Bridge services work

The AEM Forms Central Migration Bridge services are AEM OSGi Services that invoke the Central Agent executables. The services are written in Java (like all OSGi Services) and make a call to Java's Runtime.exec() method in order to invoke the Central Agent executable files which still reside in the Central installation directory. The Central INI files that were installed are still used. Logging has changed and is no longer appended to a common file by default but instead a separate log is maintained for each Central Agent invocation. Logging changes are discussed in more detail later on.

The Central service does not need to, and should not, be running. Instead, AEM core services like AEM Workflows, Apache Sling controls the processing of incoming transactions.

The various input parameters are passed to the Central Agent executables as command line parameters (e.g. –z, -aii, -arx,-atf, etc.). Input document objects are written to a temporary file and then the names are passed in to the Central agents as command line parameters. Temporary file names are generated for the output parameters and they are also passed to the Central Agent executables as command line parameters. Lastly, any miscellaneous parameters such as the ini file location are also passed in to the Central Agent executable by the Central Migration Bridge services.

Each Central Migration Bridge service has an "Other Command Line Options" parameter that overrides the standard parameters that the Migration Bridge services generate and pass to Central's Agent executables.

Unless overridden by command line options, the results of the Central Migration Bridge service operations are a series of documents and a Central Result object (which is a composite of all the result documents). The resulting documents objects contain the contents of what is, under Central, the output file, the log file, the response file and the trace file. The document objects do not have a content type set, if the content type is required, it may be set using setContentType method

The centralDataAccess service in detail

While the other Central Migration Bridge services leverage existing Central executables (and therefore are documented in their associated Central documentation), the centralDataAccess service is completely new and warrants a more detailed explanation of its workings.

The centralDataAccess service is designed to convert key values in the incoming field-nominated data stream – values such as job parameters, email addresses, printer names to XML. It is not intended as a general field-nominated data stream conversion utility. It does not implement all the possible field-nominated commands (or even most of them). It implements knowledge of a few simple commands: ^field, ^global, ^file and ^job. It parses these commands and makes their values available as an XML document.

Here is an example .DAT file:

```
^job memo -afp"C:\forms" -alp"C:\logos"
^symbolset 108
^page 1
^field MEMO_TO
Pat Brown, Manager, Purchasing
^field MEMO_CC
Kelly Green, Manager, Accounts Receivable
^field MEMO_FROM
Chris Black, Chief Financial Officer
^field MEMO_DATE
January 31, 1999
^field MEMO_SUBJ
Budget Meeting
^field MEMO_MSG
The meeting scheduled for Monday, February 19th at 2:00
is rescheduled to Tuesday, February 20th at 3:30.
If this conflicts with other activities on your calendar,
please let me know immediately.
```

Here is the resulting XML document after processing this .DAT:

```
<?xml version="1.0" encoding="UTF-8"?>
<File>
     <jobs>
         <job>memo</job>
         <job>-afp"C:\forms"</job>
         <job>-alp"C:\logos"</job>
     </jobs>
     <fields>
         <MEMO_TO>Pat Brown, Manager, Purchasing</MEMO_TO>
         <MEMO_CC>Kelly Green, Manager, Accounts Receivable</MEMO_CC>
```

```
        <MEMO_FROM>Chris Black, Chief Financial Officer</MEMO_FROM>
        <MEMO_DATE>January 31, 1999</MEMO_DATE>
        <MEMO_SUBJ>Budget Meeting</MEMO_SUBJ>
        <MEMO_MSG>The meeting scheduled for Monday, February 19th at
2:00is rescheduled to Tuesday, February 20th at 3:30. If this
conflicts with other activities on your calendar, please let me know
immediately.</MEMO_MSG>
    </fields>
</File>
```

After processing this DAT file a user can read these values easily using any XML library.

The centralDataAccess service can also be used to extract information such as email addresses to be passed as parameters to methods of the Email service or printer information to be passed to the print method of the SendToPrinter service.

Comparing AEM forms Central Migration Bridge with Central

There are some significant differences between Central applications and AEM applications utilizing the Central Migration Bridge; this section details these differences and various considerations to help plan a migration.

- **No Central Job Management Database.**
  The Job Management Database (JMD) does not exist in the Central Migration Bridge. In Central it serves several different purposes:
  - Job and task definition – this is handled by the workflow design capabilities in AEM Workflows.
  - Printer definition – the target device will need to be stated in the process design or passed in via the data file.
  - Managed Memory – this feature is not included in the Central Migration Bridge. This feature dates back to a time when printer memory was limited and bandwidth speeds were low. It is no longer required with modern printers.
- **No Central Log file.**
  When Central launches an agent the log file from that agent is appended to the Central log. This means that there is one place to find all Central error messages. Under AEM forms, each time a Central Bridge service is invoked; it creates its own log. This means there is no common location where all logs are sent.

  As mentioned earlier, the –all command line option can be placed in the "Other Command Line Options" parameter when invoking a Central Migration Bridge service in order to redirect the output to a common location however be aware that due to the multithreaded nature of AEM forms the log output from multiple Central Migration Bridge services may be interleaved in the common log file unless the multi-threaded nature of AEM forms is inhibited.

  When debugging a process that uses the Central Migration Bridge services, the first step is usually to add a –all command line option in the "Other Command Line Options" parameter of that service in an effort to capture the log file. The log file will typically contain a message that points to the cause of the problem. Without specifying the –all command line options, any exception that occurs will redirect the control flow of the process down the exception path and any log file will be lost.

- **AEM forms processing is multi-threaded.**
  Central processes one incoming job at a time so it is possible to use fixed names for temporary files. Each job completes before the next one begins, allowing the same filename to be reused by each job. Under AEM forms multiple jobs may run at the same time, so temporary files must be uniquely named. The use of AEM forms docmanager document objects is recommended, where possible.

- **AEM forms Central Migration Bridge (Output Central Service) triggering mechanisms are different.**
  Central accepts jobs in other ways besides the Central collector directory such as via named pipes and through print queues. In AEM forms primary mechanism to access Central Migration Bridge is via directly calling API methods. It is also possible to create watch folders using AEM Workflows and launcher functionality.

- **AEM forms Central Migration Bridge (Output Central Service) does not provide any**

auxiliary utilities for managing the input queue.
Central provides several utilities that allow an administrator to display and manipulate jobs in the Central queue. No corresponding utilities exist in AEM forms for the Central Utilities (jfControl, jfc, jfjmd, jfkick, jflisten, jfp, jfq, jfrm, and jfstat).

- **AEM forms Central Migration Bridge (Output Central Service) performs exception handling.**
  If any errors occur in the execution of a Central Migration Bridge service they will generate an AEM forms Output Central exception which must be handled by the process invoking the Central Migration Bridge service. Unless overridden by a –arx command line option, there will not be a response file if service detects an error during a Central Bridge service operation.

- **AEM provides email capabilities.**
  There are no Central Migration Bridge services for sending emails because existing AEM mechanisms facilitate this already.

The AEM forms Central Migration Bridge is designed to allow existing Central customers to leverage the benefits of moving to AEM forms but the Central services still have some characteristics that are different than typical AEM forms services. Experienced AEM developers should not be caught off-guard by these characteristics:

- **None of the Central Migration Bridge services has any knowledge of the AEM forms repository.**
  Any repository resources that are not passed in as input parameters must be written to disk prior to invoking a Central agent. The filenames of the temporary files must be passed to the Central agent through the usual means (e.g. via an input file or via a command line option).

  It is generally intended that Central assets used by the Central Migration Bridge will reside on the local file system because that is where Central traditionally stores them and doing so will reduce the amount of conversion work that needs to be done to access the assets from the Central Migration Bridge services. For example, if the incoming data stream references any external resource, such as using a
  ^form, ^file or ^graph command, then that external resource must be accessible via the local file system they cannot reside in the AEM repository.

In many simple scenarios though, it may be possible to store assets in the AEM repository, e.g. if all the assets used by a Central Migration Bridge service were provided to it via AEM forms DocManager document objects that are passed to it through input parameters. Storing assets in the AEM repository will simplify the resulting process.

Typical Usage Scenario

This section outlines the steps in a typical usage scenario for the Central Migration Bridge services.

1. Understand your current Central application.
   - Understand how your current JMD works, including what steps are tied to which jobs, which Central agents are invoked and what command line options they are invoked with.
   - Understand how your current forms work, including what inline commands are used to reference external files, e.g. \form, \graph and \subform.
   - Understand what your current data stream looks like and how it functions, including the inline commands above and Dynamic Merge commands that reference external files, e.g. ^file, ^form, ^graph, ^passthru, and ^shell.
   - Understand which assets are required (.mdf, .tdf, etc.)
   - Understand how interfacing applications invoke Central
   - Understand what printers Central prints to and the network protocols use to access them

2. Install AEM forms onto a machine where Central is already installed.
   - On a new machine, this means installing Central first followed by AEM Forms.
   - Alternatively, it could mean installing AEM forms into your existing Central environment. In the latter case, make sure the existing Central Environment meets or exceeds the AEM minimum system requirements. See the https://dev.day.com/docs/en/cq/current/deploying/technical_requirements.html document for the AEM technical requirements.

3. Make your Central assets available from AEM.
   - On a new machine, this means copying all assets on to the new machine.
   - On an existing machine, it means making sure that the user that the AEM runs under has permissions that allow access to the directories containing the existing Central assets.

4. Create workflows using AEM Workflow editor to replace the JMD.
   - Develop and test your workflow to replace your JMD.
   - Create a Launcher to create a watch folder to invoke the workflow.
   - If printing, your workflow will include a call to print API of SendToPritnter Service. Make sure AEM has access to the printer and that the SendToPrinter service has the correct parameters to access correct network printer protocols to address the printer.

Central Migration Bridge Samples

Adobe provides a sample to demonstrate how to utilize the Central Migration Bridge service. See http://helpx.adobe.com/aem-forms/6-1/configuring-watch-folders-central-migration.html that explains how to use the AEM Central Migration Bridge service with a watch folder.

4. Forms

This chapter will explore topics related to migrating from Central forms to AEM forms XFA Forms, but does not attempt to be a tutorial or guide to the process of designing forms with AEM forms Designer. The topics explored in this section include:

- How to directly import your existing Central forms, originally created with Output Designer, into AEM forms Designer; and, a discussion of the goals and limitations of the import process. See the section called "*Importing Output Designer Forms*".

- Information related to designing basic forms, dynamic forms, and handling multi-part Central forms. See the section called "*Basic Forms*", the section called "Dynamic Forms", and the section called "Multi-Part Forms" respectively.

- A discussion of a capability with XFA forms to create partial forms, known as fragments, which can be assembled into whole forms. See the section called "*Fragments*".

- The ability in XFA forms to store arbitrary information in the form, in a manner equivalent to variables or **docvars** in forms created with Output Designer. See the section called "*Form Variables*".

- The mechanism in XFA forms to produce the page count information (e.g. page 3 of 5) that commonly appears on generated documents. See the section called "*Page Counts*".

Importing Output Designer Forms

AEM forms Designer has the capability to import forms originally created with Output Designer. The Output Designer form files, with `.ifd` file extensions, can be opened from the AEM forms Designer standard Open dialog by selecting the Output Designer Form (`*.ifd`) display option from the Files of type drop-down list. Detailed instructions on the steps required to import an Output Designer form are described in the "Importing Adobe Output Designer Form Files" section of the AEM forms Designer Help.

Although the AEM forms Designer user-interface exposes the capability to import Output Designer forms via a **File > Open** operation, it is important to remember that Output Designer form files are imported into the XFA format – in other words, AEM forms Designer does not provide a capability to save forms in an Output Designer compatible `.ifd` format.

Import Goals and Constraints

In order to successfully import Output Designer forms into AEM forms Designer, the following minimum requirements must be met:

- There must be a functioning installation of Output Designer present on the same system as AEM forms Designer. The import process utilizes components of the Output Designer installation.

- Best results will be achieved when using AEM forms Designer ES and Output Designer 5.7.

- If the Output Designer forms have dependent resources, such as image files, those resources must be present in the locations referenced by the Output Designer forms.

- The presentment target `.icf` configuration file required by the Output Designer should be available within the Output Designer installation.

The import capability does not import interactive form features (such as field validations) that may be present in form files created with older versions of Output Designer.

The overall intent of the AEM forms Designer capability to import Output Designer forms is to jump-start the process of migrating existing forms to AEM forms Designer, not to replicate the precise behavior of existing forms. However, AEM forms Designer will attempt to migrate features of Output Designer forms, as follows:

- Form objects (e.g. text, graphics, fields, barcodes, etc.) are migrated to equivalent XFA form objects.

- Grouped objects are migrated to subforms.

- Foundation pages (`JFMain` pages) are migrated to AEM forms Designer master pages.

- Fonts used in the original forms are recognized, along with Output Designer font mappings. AEM forms Designer will provide for additional font mapping during the import process.

- Subforms are migrated to AEM forms Designer subforms, with consideration of the original subform types, relationships, and dependencies on foundation pages.

- Preamble information is processed, in order to assist in the migration of subforms.

The following sections elaborate on the aforementioned aspects of the import capability.

Form Objects

AEM forms Designer supports the same basic form objects as Output Designer: static content such as text, lines, boxes, circles, images, and barcodes; dynamic content such as text fields, radio-buttons, checkboxes, barcode fields, and image fields. Beyond these individual objects, there are equivalent features for groups, subforms, and other equivalent capabilities that are discussed in the following sections.

In general, AEM forms Designer provides a richer set of form objects, with a wider range of properties and features, than Output Designer. However, the remainder of this section describes any notable issues with importing individual form objects from Output Designer forms into AEM forms Designer.

Global Fields

It is not uncommon for Output Designer forms to have more than one field with the same name, possibly in different subforms, where some (but not all) of these fields are configured as global fields. Global fields in XFA forms behave differently than global fields in an Output Designer form, as described in the section called "*Global Fields and Global Data*". It is not permissible for a XFA form to have multiple same-named fields with varying global/non-global states within the same form
– the fields must either be all global or non-global.

Tables

AEM forms Designer does provide a capability to create table objects. However, table objects within an Output Designer form are not migrated to XFA table objects. Instead, the fields that comprise the cells of the imported table are migrated to individual fields, and the graphical aspects of the imported table are migrated to a series of lines and boxes.

For more information on tables in XFA forms, see the section called "*Tables*".

Graphics Formats

AEM forms Designer will import logo objects (images) represented in the most common formats (e.g. `.tiff`, `.bmp`, `.gif`, `.png`, `.eps`, `.jpg`), but it does not support all Output Designer image formats (e.g. `.lgo`, `.pcx`, `.hgl`).

Grouped Objects

When importing an Output Designer form into AEM forms Designer, any form objects that are grouped together are migrated to subforms in the resulting form. These subforms are given the same position, dimension, and name, as the corresponding groups from the Output Designer form.

In both Output Designer and AEM forms Designer, grouping form objects together is often performed simply to make operating on these objects more convenient. AEM forms Designer does have the notion of a group, distinct from the notion of a subform. Grouping a number of objects together in AEM forms Designer does not automatically create a new subform, although both Output Designer and AEM forms Designer permit you to select multiple objects and request that a new subform be created to enclose the select objects.

Regardless, the import processing of Output Designer forms considers groups to be significant form features, and migrates any groups to subforms in the resulting XFA form.

Foundation Pages

Dynamic forms created with Output Designer require at least one full-sized page to be defined, known as a foundation page or `JFMain` page. Subforms within an Output Designer form can also be associated with specific foundation pages. When Central generates output, it instantiates as many foundation pages as required to hold the range and variety of subforms.

Foundation pages also determine physical page attributes, such as page size and orientation. Common document features, such as page header and footer content, are often designed onto foundation pages rather than subforms.

Each foundation page of an Output Designer form is imported into AEM forms Designer as a master page in the resulting XFA form. Any associations between subforms and foundation pages are also migrated so that the subforms within the resulting XFA form will be associated with the corresponding master pages.

Fonts

AEM forms Designer will detect the fonts used in Output Designer forms, and can honor any font mapping defined for Output Designer with a jfontmap.ini file.

When importing an Output Designer form, for each font encountered that is not present on your system and is not already mapped via `jfontmap.ini`, AEM forms Designer will present you with a dialog where you may select a substitute font. For a more detailed discussion of fonts and font mapping with AEM forms, see the section called "*Font Handling*".

Output Designer, regardless of which presentment targets a particular form was designed for, provides a special font setting that can be used with any text field: the `*NOPRINT*` font setting. Any field that uses the `*NOPRINT*` font setting will, as the name implies, not print the contents of the field. AEM forms Designer does not provide a `*NOPRINT*` font setting, but it does provide several different ways to conceal content. For instance, a field may be defined to only appear on-screen, only when printed, or be defined as hidden entirely (though the field remains visible within AEM forms Designer it will not be visible on any generated output). This property of a XFA form object is known as the **presence** property. Any fields within an Output Designer form that use a `*NOPRINT*` font setting are imported as fields with the presence property set to hidden.

Subforms

Dynamic forms created with Output Designer may contain subforms that represent header, trailer, or detail types of subform content.

Subforms are instantiated and arranged by Central according to rules and relationships defined within Output Designer, such as indicating the relationship between the header, detail subforms, and trailer, representing tabular information within a form, or a specific sequencing of subforms specified by indicating that each subform has a parent subform that must appear before it.

XFA forms have equivalent subform capabilities, but they are not expressed in the same way as subforms created with Output Designer. For instance, within Central forms, subforms can be specified to occur in a specific sequence, but not by indicating that the preceding subform is a parent of, or encloses, the current subform – a parent/child relationship between subforms in XFA forms expresses that one subform is actually contained within another subform. Unlike Central forms, XFA forms may have subforms actually nested within other subforms. Therefore, when importing a Central form, AEM forms Designer attempts to migrate the subform characteristics of the original form to an equivalent XFA subform definition.

One example of an Output Designer subform characteristic for which there is no AEM forms Designer equivalent, is the capability to declare that a subform must reserve an amount of space on the foundation page large enough to encompass itself and other subforms, plus an additional arbitrary amount of space. This mechanism permits a form to ensure that a subform will not be separated by a page break from a specified range of following subforms. XFA forms provide equivalent functionality by different means, such as specifying that a subform must be kept on the same page as the following subform, or that a subform cannot be split across pages.

Preamble Handling

In both Output Designer and Central, dynamic subform behavior is largely dependent upon the instructions within a form's preamble. Output Designer automatically creates a number of different custom variables, containing preamble instructions, derived from the subforms designed within the form. These default preamble instructions, given that they are generated by Output Designer itself, are interpreted during the import to AEM forms Designer. By considering the preamble, AEM forms Designer will attempt to replicate the behaviors of the original subforms, thus producing a form that will generate either the same, or similar, output.

Beyond the automatically generated preamble, Output Designer also permits the creation of additional preamble instructions that may augment or override the default preamble. Preambles can also be specified as part of a Central job definition, or from within the data associated with a Central job. The ability of AEM forms Designer to leverage preamble instructions decreases with custom preambles, and clearly preamble information that may have accompanied a Central job or data stream is unavailable to the AEM forms Designer import process.

Working with Imported Forms

As described earlier, the AEM forms Designer import process operates according to a number of assumptions. The import process strives to produce a form that will generate a document very similar to the original Output Designer form.

Consider the following two examples intended to illustrate what is meant by unstructured XML data, versus structured XML data:

Example of unstructured XML data

```
<data>
  <vendor_code>1001</vendor_code>
  <vendor_name>A1 Business Products</vendor_name>
  <vendor_address>234 Second St., Anytown, ST</vendor_address>
  <billto_name>John Doe</billto_name>
  <billto_address>15 Fourth St., Anytown, ST<billto_address>
</data>
```

Example of structured XML data

```
<data>
  <vendor>
    <code>1001</code>
    <name>A1 Business Products</name>
    <address>234 Second St., Anytown, ST</address>
  </vendor>
  <billto>
    <name>John Doe</name>
    <address>15 Fourth St., Anytown, ST<address>
  </billto>
</data>
```

The example unstructured XML data represents data in a manner very similar to the Central field-nominated data format, with uniquely named data items. Such an equivalent field-nominated data file might look like the following:

Example field-nominated data

```
^field vendor_code
1001
^field vendor_name
A1 Business
Products
^field vendor_address
234 Second St.
Anytown, ST
```

In contrast, the structured XML is able to reuse the names of data items that represent the same type of information (e.g. a name, an address) and use structural information (the `vendor` and `billto` elements) to distinguish between information about the vendor versus the buyer.

One consequence of import process goal is that the import process creates a hierarchy of subforms intended to replicate the behavior of the original form. This hierarchy does assume that incoming data stream will also be structured in a similar fashion. If so, then the incoming data should merge with the form based on the field names. If the incoming data stream is unstructured or is structured but in a different way than the form, then the easiest way to address this is to explicitly bind your fields to the incoming data by setting the field's data binding property to the appropriate incoming XML element.

The "import fields only" checkbox

Under previous versions of AEM forms Designer, when you imported an Output Designer form, AEM forms Designer went to great lengths to make sure that the form continued to work with an unstructured XML stream by adding extra subforms and extra data transformation instructions into the imported form. Experience has indicated however that for most people who convert forms from Output Designer to AEM forms Designer this extra effort was unnecessary. Since the incoming data stream had to be converted from field-nominated to XML, converting it to a structured XML that mimicked the form's structure was no more trouble than creating unstructured XML. Converting to a structured XML however meant that the extra subforms and extra transformation instructions had to be removed from the form manually using the XML source view. This was not desirable so the import process was modified to assume that the incoming data stream was being remodeled into structured XML.

This change in behavior may prove disruptive for someone in the middle of converting a large number of forms, so a mechanism was included to tell AEM forms Designer to perform the conversion in the same way as older versions of AEM forms Designer. This is the "import fields only" checkbox on the File Import Options dialog. If this field is checked then the import process assumes that the incoming XML is structured or that the user will explicitly bind the fields. This is the default setting and is the recommended setting. If this field is unchecked, then the import process functions as it did in previous versions. It adds in extra subforms and extra data transformation instructions.

Basic Forms

Output Designer and Central process two largely distinct types of forms: **static forms**, and **dynamic forms**.

Static forms generally aren't comprised of many subforms – all of the fields and other form objects are placed in specific locations across as many pages as required to hold the potential maximum range of data. Regardless of how much, or how little, data is combined with the form, the form will look largely the same. In this way, static forms are equivalent to manual paper forms.

Dynamic forms are all about supporting a range of potential document content and layout. During the process of designing a dynamic form, the form is decomposed into its component regions of content: the subforms. Depending on the requirements of the data, a unique document will be generated that satisfies the requirements of that particular range of data. Because the content within a dynamic form is elastic, and the number and types of pages are not static, Output Designer requires that the underlying physical page requirements of the form be determined via a mechanism known as **foundation pages** (also known as **master pages** in AEM forms Designer).

XFA forms do not significantly differentiate between static and dynamic forms. All forms require master pages, and all forms make use of subforms. What determines whether a XFA form behaves like a static form, or a dynamic form, is how the individual subforms are defined – whether the subforms are located at fixed positions and not configured to conditionally appear based on a relationship with a data source, or whether the subforms are configured to appear based on the requirements of the data and flow into a location on a page determined at the time of output generation.

AEM forms does expose the notion of a static versus dynamic form in relationship to creating PDF forms for Adobe Acrobat. When designing a form for use in Adobe Acrobat, AEM forms Designer permits you to state whether the form should behave as a static or dynamic form. However, when designing forms for use with AEM forms Output Service, the distinction between static and dynamic forms is largely irrelevant.

Thankfully, the lack of this distinction does not raise the level of effort required to design simpler forms in AEM forms Designer. Simple forms can be designed in AEM forms Designer without being encumbered by features required only when designing more advanced, more dynamic, forms.

Multi-Part Forms

Output Designer provides a capability to design forms that replicate multi-part forms that are printed in impact printers or filled manually. The form has a number of layers, or parts, that may each display or hide a portion of the data. There is one set of data, and that data appears in varying degrees, on all parts of the form.

By default, a new form created with Output Designer, is a single-part form. A form may be defined as either a multi-part form, or multi-part sorted form, by selecting **Format > Template Design > Template Properties**, and choosing the desired option in the **Collate** area of the dialog box.

XFA does not provide a multi-part form capability equivalent to the behavior of Central. However, it is certainly possible to create a XFA form that has multiple pages, and utilizes data binding or scripting to replicate the data across the pages.

Dynamic Forms

As described in the section called "*Basic Forms*", AEM forms Designer does not significantly distinguish between static forms, and dynamic forms. Hence, while the form capabilities described in the following sections are enclosed within a major section entitled "*Dynamic Forms*", these capabilities can also be utilized in simpler forms that don't appear to exhibit the characteristics commonly associated with dynamic forms.

Master Pages

Master pages are a feature of XFA forms similar to foundation pages in Output Designer (as briefly described in the section called "*Foundation Pages*").

In Output Designer, foundation pages are only necessary when working with dynamic forms, and they can only contain static content (no fields). In AEM forms Designer, all forms have at least one master page, and master pages are not significantly restricted in the type of objects they may contain. Master pages may contain fields, and even subforms. However, this capability for placing objects on master pages should not be abused; even when designing simple static forms, resist the temptation to design the form within master pages, and only use master pages for information commonly found in document headers and footers.

Common form requirements such as a page header or footer that contains calculated data (e.g. today's date, page number) or information retrieved from a data source are satisfied by placing fields on master pages. For more information on page numbering, see the section called "*Page Counts*".

Similar to foundation pages, master pages also determine physical page attributes, such as page size and orientation.

The content of a XFA form is placed onto the master page at a position, and within a region, defined by an object known as a content area. By varying the position and size of a content area, the content of the form may be moved and constrained. Master pages may have more than one content area, allowing the content of a form to flow into more than one region on the page, similar to multi-column layouts.

Master pages may be defined as automatically repeating in order to completely enclose all of the form content for a given document, or may be defined to occur a specific number of times.
Depending on this definition, when a master page is full, it will automatically repeat, or the remaining form content will resume on the next available master page (assuming multiple master pages were designed). Form objects may be split across page boundaries, or may be configured as unbreakable.

Master pages may be grouped into a series of master pages with a particular ordering, known as a **master page set**. These sets can also be grouped.

The range of capabilities present in XFA master pages is significantly greater than foundation pages in Output Designer and Central. For more information on master pages, and how to use them, consult the AEM forms Designer product documentation.

Preambles

As described in the section called "*Preamble Handling*", both Output Designer and Central, achieve dynamic subform behavior by leveraging the instructions within a form's preamble. Output Designer automatically creates a number of different custom variables, containing preamble instructions, derived from the subforms designed within the form.

XFA forms do not provide a mechanism similar to preambles. Instead, XFA forms provide a rich set of capabilities for achieving the same goals. The following list summarizes some of the common tasks performed by preambles, and the equivalent mechanism in XFA forms:

- Preambles detect references to fields that are not present in the current subform and switch to the appropriate subform containing the requested field. This behavior is a consequence of the limitation inherent to Central forms where only one subform is active at any given moment and data must fully populate a subform before moving to the next subform. XFA forms do not have these limitations, and also benefit from robust data-binding mechanisms. XFA forms are capable of automatically detecting when a different subform must be instantiated, or a new instance of a subform, without restricting access to other parts of the form.

- Preambles express how newly instantiated subforms are positioned relative to previous subforms. XFA forms permit subforms to be positioned in a specific location within an enclosing subform, or arranged in sequence, and flowing in a specific direction.

- Preambles describe how much space is potentially required on the current page, or must be reserved, for a subform and its potential subsequent subforms. XFA forms provide a rich set of properties on subforms that determine whether a subform may be split across a page boundary, or whether it must be kept on the same page as the subsequent subform.

- Preambles determine the header, detail, and footer behaviors that are very common in forms. XFA forms permit subforms to be nominated as leading or trailing a detail subform (thus the XFA terminology leader and trailer instead of header and footer). In addition, XFA forms provide a powerful table capability where subforms can be arranged in a tabular layout.

- Preambles provide a mechanism similar to scripting, with the ability to perform conditional logic, test conditions, and format data. XFA forms expose a rich set of properties and events on form objects that can be augmented with scripts defined in either the **JavaScript** or **FormCalc** language.

Expandable Objects

Forms created with Output Designer may leverage a capability where form fields may expand vertically to accommodate a range of data that would not otherwise fit within the field. By specifying in Output Designer that a field can expand, any data that would otherwise overflow the field is continued onto as many instances of a similar field in another subform as required. These additional subforms typically appear below the original subform containing the original expandable field enclosing the first line of data. A common use case for this capability is where a particular column within a table may contain free-form text associated with a line-item, or row, of the table. All of the descriptive text would appear within a visually taller table cell contained within the row, and the overall height of the row expands such that successive rows of content are properly located below.

The Output Designer approach works by repeating a particular subform for each line of content that overflows the original expandable field. However, while this approach works for the above simple and common scenario, it has two notable limitations. First, only one field representing a column within a subform representing a row can be configured as an expandable field, and the first field that begins to overflow produces the additional subforms. Once these additional subforms, representing the overflow area, are produced, Central's processing is unable to return to the original subform to handle additional expandable fields. Further, any additional data intended for the original subform that occurred after the overflowing data will be lost. Second, expandable fields can only expand vertically. There is no capability for fields to expand horizontally.

XFA forms permit objects, such as fields and subforms, to be defined as expandable in either a vertical or horizontal dimension. AEM forms Designer provides, via the Layout palette, the ability to specify the minimum size of an object and whether the object should expand vertically or horizontally to accommodate its content, and in which directions the object should expand.

This ability for objects on XFA forms to expand either vertically or horizontally is a first-class feature of the objects themselves; it does not require, unlike Output Designer, that the additional content be accommodated by fields and subforms designed to capture the overflowed content; and therefore, XFA forms do not exhibit the limitations or side-effects of Central forms described above.

As an expandable object on a XFA form increases in size, the decorative attributes of the object (such as the border) will also automatically adjust. Depending on whether the object has been anchored to a particular location on the page, or whether the object floats alongside the other objects in the form, the object's expansion may overlap other objects or cause the layout of subsequent objects to adjust accordingly. Eventually, an object may grow so large that it may need to be split across pages, and

AEM forms Designer allows the object to be configured so that its border will appear open or closed on either side of the page break. A subform may also be configured to permit, or disallow, the splitting of its content across pages.

Field Overflow

Related to the previous discussion of expandable objects (see the section called "*Expandable Objects*"), is the topic of how oversized content is handled within fields defined with a fixed size.

Consider the scenario where a single-line field, such as a field intended to represent a telephone number, is configured to hold only seven digits. When such a field is given more than seven digits of data, such as a telephone number that includes an area or country code, an overflow condition occurs.

In Central, by default, special handling is provided for single-line fields such that data is permitted to extend beyond the outer boundary of the field. In the case of the too-long telephone number, all of the digits would appear to extend beyond the edge of the field. In addition to this default behavior for single-line fields, any field object in a Central form may be configured, via preamble processing, with instructions on how to respond to an overflow event. Multiple-line fields may also take advantage of preamble handling of overflow conditions, but by default multiple-line fields will wrap content onto successive lines within the field until the field is filled with content. The handling of any remaining content is dependent upon the current **reformat** mode of Central (consult the Central documentation for more information on reformat processing).

XFA forms do not exhibit different behavior for single-line fields versus multiple-line fields, except for the expected behavior that multiple-line fields will automatically wrap content onto successive lines. Fields are either designed with a fixed width and height, or may be designed to expand in width or height. A field with a fixed dimension will eventually truncate any content that does not fit, whereas an expanding field will extend to accommodate the content. Fields in XFA forms do not provide a mechanism similar to the preamble-based overflow handling in forms created with Output Designer.

AEM forms Designer also provides a type of field, the image field, intended to present an image rather than textual content. Image fields can be configured to scale the image to fit the dimensions of the field (optionally preserving the aspect ratio of the image), or render the image according to its intrinsic dimensions.

Tables

Tabular layout of data is a very common feature of forms, and both Output Designer and AEM forms Designer provide features for designing tables. However, often tabular layout is accomplished in Output Designer and Central by using a series of subforms, rather than a table object, often because subforms provide additional features (such as pagination) and allow the content of a table to vary by selecting from a variety of subforms.

Recognizing that tabular layout, for all but the simplest of forms, often requires functionality commonly associated with subforms, AEM forms Designer exclusively utilizes subforms to produce tables, yet provides a rich user-interface suitable for both simple tables and complex tabular layout.

Tables in XFA forms provide the common features of headers and footers, with control over the repeating of headers and footers across page breaks. The content of a table cell can be any form object, and may be bound to a data source with a varying number of rows dependent upon the range of data. Additional features of XFA tables include:

- The number of rows can be fixed, or bound to a range of data.
- The content of a table cell can be any form object, including a subform, or a nested table.
- The table content can be subdivided into sections with independent headers, footers, etc.
- The rows or columns of the table can be configured to be automatically equally sized.
- The layout of a table can be configured to automatically break based upon a scripted condition.

For more information on designing tables in AEM forms Designer, consult the AEM forms Designer product documentation.

Floating Fields

Output Designer provides a feature known as **boilerplate fields** that permits the content of an otherwise static region of text to contain regions of variable content that will be populated from data, and the layout of the surrounding content automatically adjusts to accommodate the variable content.

XFA forms provide an equivalent mechanism, known as **floating fields**, where a field can be inserted within the content of a region of text on a form, the field can be populated with data when the document is generated, and the layout of the surrounding text is adjusted. The underlying mechanism that XFA forms use to accomplish this is to embed a specific unit of XML markup within the text content of the form object. This mechanism can also be used to reference and substitute values into the data that will be combined with a form (see also the section called "*Embedded Field References*").

Calculations

AEM forms Designer is a full-featured form design application that can be used to create interactive forms, forms designed solely for generating output documents, or any combination in between. Hence, features commonly associated with interactive forms, such as scripting and fields that contain calculated values, are available for use within document generation scenarios. When a form is processed, along with any associated data, any dependent scripting and calculations contained within the form are automatically executed.

In comparison, Central field-nominated data files and form preambles may use a simple command language with conditionals and common operations known as **calculation expressions**. These calculation expressions are very useful, but they are not a complete scripting solution. The XFA approach leverages the existing scripting capabilities of XFA forms, using either JavaScript or FormCalc scripting languages. It is worth noting that a set of functions is provided within the XFA FormCalc scripting language that are very similar to the functions available to calculation expressions.

Fragments

The field-nominated format used by Central provides mechanisms for referencing multiple subforms from within a single data-stream. In this way, a document can be constructed by assembling parts of different forms together. The field-nominated `^subform` command permits the data file to call out to another subform within either the current, or a different, form file.

XFA provides a similar mechanism known as fragments. Where Central provided this functionality by using a `^subform` command from within a field-nominated data file, XFA provides the ability to create, manage, and reference reusable fragments from within the rich user- interface of AEM forms Designer.

With AEM forms Designer, any collection of form objects, including script objects, can be turned into a fragment, stored in a library of fragments or within a saved form file, and recalled from another form. Fragment libraries are storage locations, either on the local system or a shared network location, where fragments are stored. Multiple libraries can be created to aid in the organization, sharing, and reuse of fragments.

The AEM forms Designer user-interface for fragment libraries is a palette equivalent to the Object Library palette. Fragments are available from the fragment library palette to be inserted into the current form.

Inserting a fragment into a form provides the same visual feedback and appearance as if the individual objects had been manually inserted into the form. However, when inserting a fragment, a reference to the storage location of the fragment within the fragment library is retained within the form. In this way, any changes made to a fragment will be automatically reflected when the referencing form is next opened in AEM forms Designer, or processed by the AEM forms Output Service

Fragments in a form can also be disassociated from their original definition within a fragment library, allowing a fragment to be effectively copied into a form and isolated from any future changes that might be made to the original fragment in the library.

As briefly mentioned above, a fragment can be created within a form and simply stored as part of a form, rather than stored within a fragment library. While such a fragment will not be available from within the fragment library palette, AEM forms Designer also provides a mechanism to insert fragments located within other XDP form files. This capability is conceptually very similar to the Central `^subform` command that directly references a subform stored within another form file.

For more information on creating and using fragments, consult the AEM forms Designer product documentation.

Form Variables

Forms created with Output Designer may have additional name/value pairs of information stored within the form. Output Designer facilitates the creation of these name/value pairs as **Custom Properties**, whereas Print Agent exposes them via the **DocVar** dictionary.

AEM forms Designer also permits the creation of custom variables, associated with a form, by using the **Variables** tab of the **Form Properties** dialog. These variables can be accessed by referencing the named variable from within any scripts.

Page Counts

A common requirement of multi-page documents is the appearance of a running page count, along with the total page count, on the header or footer of a document. Often this will appear as "Page *n* of *m*" where *n* is the current page number and *m* is the total page count.

Obtaining the current page number is straightforward in Central, such as using the `^$page` or `\$page` field-nominated commands. Determining the total number of pages is more challenging, because the document must be first generated in order to determine the total number of pages, though this process can be automated with Central's job management database. With Central, the need to predetermine the total number of pages is due to the way that documents are constructed. In order to print a "Page *n* of *m*" on the first page of a document, the value of *m* must have already been determined because by the time the end of the document is encountered, it is too late to go back to the previous pages and subforms to populate the areas referencing the total number of pages.

XFA takes the approach of constructing an entire document as one whole entity before committing the generated document to its output format or device. In this way, scripting and calculations have an opportunity to execute in the context of a fully constructed document, and may easily query the document for its total number of pages. No multi-step processing is required to determine the total number of pages.

Individual master pages in a XFA form may indicate whether the page should be numbered, and contribute to the overall page count, or whether the master page should not be considered a numbered page. This is useful for scenarios where one or more pages, or possibly the back side of pages in a duplex document, are not intended to have their own page number. A master page may also be configured to start a new page count, or continue from an existing page count.

XFA forms also distinguish between the total count of pages (based on the accumulation of numbered master pages) and the total number of surfaces representing the actual number of physical page surfaces generated.

XFA forms retrieve information about the current page, the total number of pages, and the total number of surfaces by calling a built-in XFA method as the calculated value for a field, or from a script expression. For instance, in order to obtain the total number of pages or surfaces, a script would call the `xfa.layout.pageCount()` method or the `xfa.layout.sheetCount()` method respectively. Obtaining the current numbered page or surface is handled by calling the `xfa.layout.page(this)` or `xfa.layout.sheet(this)` method respectively. These two methods receive a reference of the calling object (`this`) to determine on which page or surface the calling object appears. Hence, it is also possible to determine on which page or surface a different object appears by passing in a reference to that object when calling these methods. This is useful for producing cross-references within a document. Note that when calling these methods, the first page is (by default) numbered `1` (one) whereas the first surface is numbered `0` (zero).

To ease the creation of "Page *n* of *m*" areas on a form, AEM forms Designer provides a "Page *n* of *m*" object in the Object Library. A similar "Sheet *n* of *m*" object is also available from the Object Library that produces surface count information instead of numbered page count information.

XFA forms also provides page and sheet values in the context of a document batch allowing for page counts to be determined outside of the context of a document and in the context of a complete batch. Some Central solutions mark document batches for use with enveloping / insertion machines. This is typically via a 2 pass process, capturing page information via preamble TRACE command in the first formatting pass, running a custom program to manipulate the batch data file based on the captured TRACE information and then doing a final format with enveloping / insertion marks on each page. These batch level values can be used to script enveloping / insertion solutions without the need for 2 formatting passes or a custom program. For more information on accessing these values see the AEM forms Designer ES Scripting Reference.

Locale Settings

A document may need to be used within a particular environment with its own language requirements, conventions for representing dates, times, numbers, and monetary values. International standards exist for categorizing the world's various collections of geopolitical and linguistic expectations around representing and interpreting such data content; commonly these individual collections are known as locales.

XFA forms leverage these standards and provide an easy way to build forms, and generate documents, that can adapt to a particular locale, or present information according to the expectations of multiple locales within the same document. For example, a document could correctly present its dates and numbers regardless of whether the document was generated in an English or French locale, or an American or British locale. A document might need to present its information simultaneously in two or more languages, or ensure that its content is always consistently presented according to its country and language of origin, regardless of the country where the document generation or viewing will occur. All of this is possible with the locale functionality of XFA forms, and XFA forms also permit their locale information to be customized for the creation of new locales, or modification of standard locales.

It is important to note that the XML standard provides a conceptually similar mechanism, known as `xml:lang`, for declaring that the content of a particular XML element is expressed in a particular language. This is similar, but distinct from XFA's use of locale information, and XFA forms do not process any `xml:lang` attribute encountered when processing an XML data source.

Within a XFA form, most objects that have a capability to contain or format data content also expose a `locale` property that can be adjusted to one of the available locale definitions. The form itself also has its own overall default locale setting, which can be selected from the AEM forms Designer Form Properties dialog. Because the content of a XFA form can be comprised of a hierarchy of objects contained within multiple nested subforms, each object defaults to inheriting the locale setting of is enclosing subform, or the overall default locale for the form. An object, or the form itself, may also choose to not constrain itself to a particular locale, and instead may choose to operate according to the locale of the system or application environment.

Central provides configuration settings for adjusting common locale features, including the grouping separator (thousands separator) or decimal point for numeric values, currency symbol, and calendar content (e.g. day names, month names). Although XFA leverages a built-in standards-based configuration for a wide variety of locales, it is possible to add your own locale information, or adjust an existing locale. Custom or modified locale information can be incorporated within the XML definition of a form itself, according to an XML markup language that is documented as part of the Adobe XML Forms Architecture Specification.

Here is an excerpt from the locale definitions automatically embedded by AEM forms Designer within a form designed for an American English locale, showing the definitions for the abbreviated day names, numeric punctuation, and currency symbols:

Excerpt of form locale information:

```
</calendarSymbols>
  <dayNames abbr="1">
    <day>Sun</day>
    <day>Mon</day>
    <day>Tue</day>
    <day>Wed</day>
    <day>Thu</day>
    <day>Fri</day>
    <day>Sat</day>
  </dayNames>
</calendarSymbols>
<numberSymbols>
  <numberSymbol name="decimal">.</numberSymbol>
  <numberSymbol name="grouping">,</numberSymbol>
  <numberSymbol name="percent">%</numberSymbol>
  <numberSymbol name="minus">-</numberSymbol>
  <numberSymbol name="zero">0</numberSymbol>
</numberSymbols>
<currencySymbols>
  <currencySymbol name="symbol">$</currencySymbol>
  <currencySymbol name="isoname">USD</currencySymbol>
  <currencySymbol name="decimal">.</currencySymbol>
</currencySymbols>
```

5. Data

Data Formats

Central is a product whose evolution began well before the **World Wide Web Consortium** (**W3C**) and the common use of XML as a data format. This history is reflected in the range of non-XML data formats that Central is capable of consuming, including comma-delimited and fixed-length record formats. However, the preferred, and most widely used, Central data format is known as the **field-nominated format**, and often referenced by its commonly used file extension as the DAT format.

In recent releases Central has provided support for XML data, by transforming XML into field-nominated format either automatically or explicitly via the XML-Import Agent. If you have prior experience using XML with Central, many aspects of that experience will apply to building solutions with AEM forms. XML is the only data format support by AEM forms Output Service, though any number of different XML-based data formats can be supported.

This section will explore what you need to know when moving from the range of data formats supported by Central, including the XML supported by Central, to the XML data consumed by AEM forms Output Service.

Central Transformation Agent

Central provides a Transformation Agent, and Visual Transformation Editor, that facilitate the transformation of one data format into another, including data formats that may not be supported by Central.

AEM forms Output Service exclusively consumes XML data, and does not provide an equivalent capability to transform non-XML data. AEM forms Output Service does provide support for leveraging XML transformations expressed in the standard **XSL Transformations** (**XSLT**) language.

Moving to XML Data

It is valuable to first understand the fundamental similarities and differences between how Central consumes field-nominated data, compared to the XML data consumed by AEM forms Output Service.

Both formats are textual, and annotate the data with markup to, at a minimum, provide each unit of data with a name that aids the process of merging the data into a form. A simple example of data in field-nominated format, and equivalent example1 in XML format, follows:

Very simple field-nominated data:

```
^field vendor_code
1001
^field vendor_name
A1 Business Products
^field vendor_address
234 Second St.
Anytown, ST
```

Very simple XML data:

```
<vendor_code>1001</vendor_code>
<vendor_name>A1 Business Products</vendor_name>
<vendor_address>234 Second St., Anytown, ST</vendor_address>
```

Beyond the syntactical differences of expressing markup in field-nominated format using the caret (^) symbol versus angle-brackets in XML, the two formats differ significantly on the meaning of their markup. The field-nominated statement `^field vendor_code` is actually a command instructing Central to apply the following data to a field known by the specified name of `vendor_code`. In contrast, the XML markup `<vendor_code>` only states that the following data shall be known as `vendor_code`.

Field-nominated format is actually a command language, not a data description language such as XML. The field-nominated format provides a range of instructions intended to be consumed by individual Central Agent processing modules. This command language typically encloses data intended to be merged with a form. This is more apparent when considering field-nominated statements such as `^page` or `^eject` that instruct Central to advance or eject a page, and clearly have nothing to do with describing data. In contrast, the XML data consumed by AEM forms Output Service is almost exclusively constrained to the task of adequately describing the data, often according to a custom schema expressed in XML Schema format.

The capability to affect the document construction and rendering process at a low-level, with commands like `^page`, undoubtedly provide the creator of field-nominated data with a significant degree of intervention, from within the data, over the generated document. However, the use of such commands does increase the degree of coupling between a set of data and a form. It can also become a challenge in solutions where the data is provided by a third-party and there is concern over the fact that the data is driving the document generation process.

There is no equivalent, in the XML data consumed by AEM forms Output Service, to the majority of field- nominated commands that are focused on affecting the document construction and rendering. For a discussion of field-nominated commands, and equivalent functionality in XFA forms, see Chapter 9, "*Field-Nominated Commands*".

DAT is flat, XML is structured

The Field-nominated format expresses data in a linear manner, with limited mechanisms to describe structure or relationships that might exist within the data. As a consequence, in many cases the order of the statements within field-nominated data is significant, and different results may be produced depending on the order.

XML data is inherently hierarchical, and each unit of data, is enclosed within tags that indicate the beginning, and the end, of each data item as an XML element. Data may also be expressed as XML attributes. The structure, relationships, and ordering of data, implied by XML data is often explicitly defined by an accompanying **schema**. While a schema can make assertions about the relative ordering of the XML data elements, such order is often insignificant. Consider the following reworking of the XML example introduced in the previous section.

Structured XML:

```
<vendor>
  <code>1001</code>
  <name>A1 Business Products</name>
  <address>234 Second St., Anytown, ST</address>
</vendor>
<recipient>
  <address>678 Fourth Ave., Anytown, ST</address>
  <name>Tony Blue</name>
</recipient>
```

In the above example, element names such as `name`, and `address` appear in two different contexts, describing the name and address of the vendor and recipient. It is the presence of the enclosing `vendor` and `recipient` tags that allow the vendor's name and address to ultimately be distinguished from the recipient's name and address. An equivalent example in field-nominated format would likely employ `^field` commands, and possibly `^group` commands, with the fully-qualified names `vendor_name` and `recipient_name` in order to adequately differentiate between the data items.

It may appear at first that the common practice of repeating element names in different contexts should be discouraged, that it creates ambiguity. One common reason for this apparent re-use of element names often stems from the use of a schema where the benefit is the ability to define a data item name or address once, and then further define the contexts in which these data items can appear. As described before, it is the surrounding context within the XML that disambiguates these data items. XML data, when used in concert with AEM forms Output Service's XML-based forms, utilizes these matching hierarchies of structural information, along with explicitly defined data-binding instructions within the form, to successfully merge the data into the form.

The sequence of name and address elements varies between the vendor and recipient to illustrate that the ordering of this information is not significant for the correct processing of this information.

XML Data is Case-Sensitive

One significant difference between using field-nominated data with Central and using XML data with XFA forms is case-sensitivity. Central data formats and forms are not case-sensitive, whereas X M L data and XFA forms are case-sensitive. Case-sensitivity is simply the accepted practice with XML technologies.

Consider that Central would consider the following two field-nominated data files to be equivalent and process them in the same way:

Lowercase field-nominated data

```
^field vendor_code
1001
^field vendor_name
A1 Business Products
^field vendor_address
234 Second St.
Anytown, ST
```

Uppercase field-nominated data

```
^FIELD VENDOR_CODE
1001
^FIELD VENDOR_NAME
A1 Business Products
^FIELD VENDOR_ADDRESS
234 Second St.
Anytown, ST
```

In contrast, depending on whether a XFA form was designed with lowercase field names or uppercase field names, only one of the following two XML data files with the matching letter-case would properly bind to the form.

Lowercase XML markup

```
<vendor>
  <code>1001</code>
  <name>A1 Business Products</name>
  <address>234 Second St., Anytown, ST</address>
</vendor>
```

Uppercase XML markup

```
<VENDOR>
  <CODE>1001</CODE>
  <NAME>A1 Business Products</NAME>
  <ADDRESS>234 Second St., Anytown, ST</ADDRESS>
</VENDOR>
```

The best way to avoid problems with the case-sensitive nature of XML and XFA forms is to establish whether your XML data, schema, and forms will use lowercase or uppercase naming, and consistently maintain that practice.

Formatting Rich-Text Data

Field-nominated data can include commands that affect the formatting of the data within a form field. These commands, known as **Inline Text Control** commands, provide capabilities such as varying the appearance of the font, defining and advancing to tab stops, and more. Data incorporating such formatting commands, or markup, is commonly called **rich-text**.

Rich-text formatting with Inline Text commands

```
^inline on
^group order
^field instructions
Deliver the order to the \b.back\b0. door!
```

Within the Central support for XML data, rich-text is not expressed using Inline Text Control commands. Instead, rich-text is expressed with a subset of XHTML markup.

Rich-text formatting with Central XHTML

```
<order>
  <instructions
    xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
    xfa:contentType="text/html">
    <p>Deliver the order to the <b>back</b> door!</p>
  </instructions>
</order>
```

AEM forms Output Service supports an extended range of XHTML-based rich-text markup, including support for a subset of CSS style attributes.

Rich-text formatting with AEM forms Output Service XHTML

```
<order>
  <instructions>
    <body xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
          xmlns="http://www.w3.org/1999/xhtml">
      <p>Deliver the order to the <b>back</b> door!</p>
    </body>
  </instructions>
</order>
```

Embedded Field References

Building on the earlier discussions of floating fields and rich-text data (see the section called "*Floating Fields*" and the section called "*Formatting Rich-Text Data*" respectively), XFA forms can also embed referenced content from within data, in a manner equivalent to variable substitutions in field-nominated format.

Field-nominated data can contain references, identified by the "`@`" (at-symbol) prefix, to the value of another field or previously defined variable. For instance, the following field-nominated data would populate the message field with the value "You saved 20 dollars":

```
^field saved_amount
20
^field message
You saved @saved_amount dollars
```

To accomplish an equivalent substitution from the data with a XFA form, special XML markup must be placed within rich-text data. The following example XML data would produce the same result as the above field-nominated example:

```
<saved_amount>20</saved_amount>
<message>
  <body xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
        xmlns="http://www.w3.org/1999/xhtml">
    <p>You saved <span xfa:embed="saved_amount"/> dollars</p>
  </body>
</message>
```

Note that in both of the above examples, it is assumed that there is a `saved_amount` field on the form (possibly a hidden field).

An optional `xfa:embedMode` attribute on the embedded reference can determine whether or not the referenced value should be inserted with or without any of its original formatting. References can a l s o be expressed as an XML id reference or URI by providing the `xfa:embedType` attribute with a value of `URI`.

One interesting capability of field-nominated variable substitutions is that they can be combined into compound references that are evaluated from right to left. In other words, given the following example, the value of the message field would be "You saved 30 dollars":

```
^field savings_tier_1
20
^field savings_tier_2
30
^field client_tier
2
^field message
You saved @savings_tier_@client_tier dollars
```

In the above example, there are two levels of potential savings that can be provided to our imaginary client, based upon the client's rating as a tier-one or tier-two client. Because the `client_tier` substitution is evaluated first, the whole substitution expression is evaluated to "`savings_tier_2`" and the value of the `savings_tier_2 field` is returned.

While this ability to combine variable substitutions provides a useful additional level of indirection, its importance in the context of the field-nominated format stems in part from the fact that this feature was established prior to advent of calculation expressions, and there are fewer overall opportunities for scripting logic. By leveraging the capability to place calculations and scripting within a XFA form, the same goals can be achieved. Consider the following example XML data:

```
<savings_tier_1>20</savings_tier_1>
<savings_tier_2>30</savings_tier_2>
<client_tier>2</client_tier>
<message>
  <body xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
        xmlns="http://www.w3.org/1999/xhtml">
    <p>You saved <span xfa:embed="saved_amount"/> dollars</p>
  </body>
</message>
```

In the above example, the same data is provided to populate the two different levels of savings and the particular level of savings that should be awarded to the client. However, the embedded `r e f e r e n c e` is to a field named `saved_amount` for which no data is supplied. In order for this example to work, a field must be placed on the form with a calculated value that determines the savings value.
The `saved_amount` field may have a JavaScript calculation such as the following:

```
if (client_tier == 1) {
  return savings_tier_1
} else if (client_tier == 2) {
  return savings_tier_2
} else {
  return 0
}
```

Unicode

The characters within field-nominated data are encoded according to the encoding scheme specified by one or more `^symbolset` commands within the data, or as part of options specified on the `^job` command. The range of characters (the **character set**) available at any particular point in the data is the range of characters that can be accessed by the encoding scheme.

Central supports a wide range of encoding schemes, including the UTF-8 Unicode encoding. Prior to the common use of Unicode, the `^symbolset` command was often invoked to access characters that were simply unavailable in any single character set. For instance, data that incorporated characters from both European and Asian languages would utilize the `^symbolset` command when switching between these languages and the corresponding encoding schemes. The `^symbolset` command also supports Unicode encodings such as UTF-8, providing a means to address the range of Unicode characters without switching among encodings within the data.

XML supports only characters found within Unicode. Unlike the field-nominated format, the entire content of an XML resource may only be encoded in a single encoding scheme. However, this should not be a limitation given the vast range of addressable characters in Unicode. XML also provides a mechanism to address individual characters by either a symbolic or numeric reference.

The encoding scheme of an XML resource is indicated by the XML declaration, appearing at the beginning of the resource (e.g. `<?xml version="1.0" encoding="UTF-8"?>`)

Binding Data to Forms

In Central, the process of combining a range of data with a form, to generate a final document, is commonly known as **data merging**. The data is merged into the fields in a manner that evokes the classical operation of merging names and addresses with mailing labels in a word processor.

With AEM forms Output Service the equivalent process of combining a range of data with a form is not known as merging; it is known as **data binding**. This difference in terminology speaks to the different nature of how XML data is bound to an XML-based form designed with AEM forms Designer.

Within XML languages, and within HTML oriented technologies on the web, the notion of making connections between units of information for a variety of purposes is commonplace. The location of a unit of information within an XML document may be connected to another location within the same XML document or within another XML document. These connections are the bindings, and the connections between a form and data are data bindings.

The process of combining data with a form could be considered as either a process of pushing data items into the form, or the form pulling from the data. In one case the data file is controlling the process, and in the other case the form is controlling the process. This distinction is important because it determines whether the control is in the hands of the person responsible for developing the data, or the person responsible for designing the form.

Central exhibits a **data-driven** approach, where the commands in the data are in control over the generated document. A form, especially a form utilizing dynamic subforms, can be designed as loose collection of subforms that will be assembled in a sequence and populated with data according to the field-nominated commands encountered in the data.

AEM forms Output Service operates primarily in a **form-driven** approach where the form contains the rules that determine how the final document will be generated. However, AEM forms Output Service also permits the designer of a form to indicate where the form should cede a degree of control to the data.

This chapter will describe these mechanisms and how they differ from Central.

Global Fields and Global Data

Globals are a mechanism for the very common requirement of having a single data value appear in more than one field on a form. Global fields can be defined with the Output Designer, and global data values can be defined from within a field-nominated data file or accompanying preamble. In either case the effect is the same: data that is recognized by Central as global is merged into the appropriate range of like-named global fields designed into the form.

Global data is handled differently when using XML data with Central. A special XML attribute, `xfa:match="many"`, is recognized on any element, and this instructs Print Agent to process the associated data value as a global data value within the scope of the current document.

When Central has finished processing the data for the current document, the global data values created with the `xfa:match="many"` attributes are forgotten, and will not be available to be merged into the successive documents generated from the same data value. Put another way, a global data value defined within the context of one document is not carried forward into the processing of successive documents. This behavior has been a challenging aspect of how Central's support for XML data is processed. The common requirement to define a global data value that is merged into a series of consecutive documents isn't accommodated by this behavior.

The behavior of global fields in XFA forms is more flexible, providing a means to merge data into multiple fields within a form, as well as mechanism to merge global data across multiple documents generated from a single range of XML data.

The first method to merge a single data value into multiple fields of a XFA form is to simply use AEM forms Designer to bind fields to the same data value. By doing this, there is nothing special, nothing global, about either the fields or the associated data value; but, the effect is equivalent to common use of globals within Central. It is not required that the name of the data value matches the name of the field, nor that the multiple fields each have the same name. By binding the fields to a data value within AEM forms Designer, the respective naming of the data value and corresponding fields becomes irrelevant.

The second method is to indicate with AEM forms Designer that one or more fields, each with the same name, are global. This will not produce the same behavior as the previous scenario of binding m u l t i p l e fields to the same data value, because AEM forms Output Service expects to find a data value, with the same name as the global fields, located outside of the current data record.

In the following example XML, data for two separate invoice documents is represented. And prior to either invoice, a data value named `date` is defined. Assuming the invoice form has been designed in AEM forms Designer such that a field named `date` is defined as global, the value of the `date` data value will be bound to the `date` field of each generated invoice.

Global data placement in XML

```
<invoices>
  <date>Sunday March 31, 2007</date>
  <invoice>
    ...data for the first invoice...
  </invoice>
  <invoice>
    ...data for the second invoice...
  </invoice>
</invoices>
```

Both AEM forms Output Service mechanisms of merging individual data values into multiple form fields produce different effects, address different use-cases, are complementary, and can be combined within a form. Choose the appropriate solution depending on whether you need to simply merge data into multiple fields within a form, or merge data into multiple fields across consecutively generated forms.

Form-Driven Data Binding

A form-driven approach to generating a document occurs when the form contains all of the rules necessary to generate a document by binding the data to the form fields and subforms.

By explicitly connecting form objects to specific data items in AEM forms Designer, the data-binding properties of the form objects are populated. Because of these explicitly designed bindings, the names of the form objects, and their location within the structure of the form, do not need to correspond to the names or position of the associated data items. The bindings determine how the form is constructed and populated with data.

For more detailed information on data-binding with AEM forms Designer, consult the AEM forms Designer product documentation.

Data-Driven Data Binding

Data-driven data-binding occurs when a form is designed to permit the data to take some control over how the form is generated from the data. Form-driven data-binding places an emphasis on defining rules that describe how the document will be constructed, and storing these rules within the form itself. In contrast, a form intended to facilitate a data-driven approach will have fewer rules defined, and will rely upon the particular sequence of data to drive the process of constructing the document.

When a form object lacks an explicit data-binding, AEM forms Output Service attempts to match the form object by name to a data item that has not yet been bound to another form object. For more information on this approach, consult the XFA Specification for information on the topics “Data Binding” and “Automatic Data Binding”.

Multi-Record Data

Often there is a need to generate multiple documents from a single data file that contains multiple records of data. Because XML has a requirement that there be only one outermost element, multiple records must be enclosed within a single, often arbitrary, element. Consider the following example XML data:

Multi-record data:

```
<batch>
  <invoice>
    …data elements for the first invoice…
  </invoice>
  <invoice>
    …data elements for the second invoice…
  </invoice>
</batch>
```

In the above example, two records of invoice data are enclosed within a `batch` element. The name of the outer element, in this case a `batch` element, is not special – it only serves to satisfy the XML requirement of a single outermost element. Also, the records within this example all represent the same type of form: an invoice. However, data representing different types of forms can be represented within a single data file, as follows:

Multi-record heterogeneous data:

```
<batch>
  <invoice>
    …data elements for the first invoice…
  </invoice>
  <purchase_order>
    …data elements for a purchase order…
  </purchase_order>
  <invoice>
    …data elements for the second invoice…
  </invoice>
</batch>
```

In order for AEM forms Output Service to distinguish between a data file intended to produce a single document, versus data intended to produce multiple documents, AEM forms Output Service must be informed of it. For this purpose Output Service provides a set of Batch operations which assume that provided data contains multiples records. In cases of heterogeneous record data, such as the above mix of invoices and purchase orders separate XFA forms for each type can be provided via Batch API.

Document Package Creation:

Consider a scenario where a document package is comprised of a letter, followed by a financial document (such as a loan), and concluded with a document intended to capture one or more signatures. These individual documents can be designed and maintained separately, and reused within different document package scenarios. Given an appropriate set of data records, mapped to appropriate XFA forms via AEM forms Output Service batch APIs, a unique document package is constructed.

A new capability in AEM forms, dynamic XDP assembly, provides a more complete solution for document package creation; the Assembler service allows a custom XDP form to be generated which can then be formatted using AEM forms Output Service. The Assembler service can insert one or more subforms at insertion points defined in XDP forms by AEM forms Designer ES and combine individual forms (including those with insertion points) into one XDP form.

Central achieves an equivalent result through the use of the `^form` command to vary between a number of different forms within one data file; page numbers will be continuous throughout the document produced by Central.

Modifying Form Objects from Data

Typically data binding is strictly the process of connecting the content of a form field with a data value, but AEM forms Output Service forms provide an additional capability to have other properties of form objects retrieve their values from the data. Subform, form fields, radio-button groups, and even static form objects (regions of text, images, etc.) can be configured to bind one or more of their properties   to the data.

One example for binding the properties of a form object to data is populating the caption property of a field from the data. A field’s caption is the visible label, or prompt, that informs the user of what information

Here a just a few simple scenarios where this capability is useful:

- Populating the items of a drop-down list from the data.
- Altering the field prompt, or caption, based on a data value.

- Altering the validation error messages of a field, based on a data value.

The ability to bind form object properties to data is activated by selecting **Show Dynamic Properties** from the Object Palette within AEM forms Designer, or by selecting **Show Dynamic Properties** from the **Tools > Options > Data Binding** dialog. Properties that support binding are then highlighted in the user-interface, and may be bound to data. Consult the AEM forms Designer product documentation for more information on “Dynamic Properties”.

6. Document Generation

Agents and Services

Central functionality is exposed by individual software components known as agents, such as the most commonly used Print Agent that generates output. Agents are invoked according to a set of rules expressed in the Central Job Management Database (JMD).

AEM forms is also comprised of individual software components, known as services that perform specific operations. AEM forms Output Service provides service operations for combining data with a form, generating output and AEM forms SendToPrinter Service for sending output to a printer, and more.

Conceptually similar to the job definitions within the Central JMD, AEM forms services and their operations can be assembled together to perform complex tasks by constructing processes with the AEM Workflows, a graphical design environment for developing workflows.

Invoking AEM Output Service

The primary mechanism that Central utilizes to receive data files, or jobs, for processing is the collector directory: a location on the file system watched by Central. Any file written to the collector directory, depending on how Central is configured, may be eligible for processing. In addition to the collector directory, data files may be submitted to Central via name pipes, a print queue, or email.

AEM forms Output Service is an OSGi service. It can be invoked in a number of different ways. A servlet or JSP can invoke AEM forms Output Service directly. Also, one can write AEM Workflow in either Java or EcmaScript to invoke Output Service.

AEM Workflow launcher provides a similar range of functionality to the Central collector directory. In AEM user creates or uses and existing Workflow. This workflow can then be bound to a location in content repository by creating launchers. Launchers in AEM can be configured to invoke a workflow whenever any node is created or modified in AEM content repository.

Identifying the forms

A typical invocation of AEM forms Output Service will include, at a minimum, two resources: a data file, and a form file. In a simple case, the form file may be specified as one of the properties on the Output Service operation, where a process to generate output has been designed within AEM Workflows. Multiple workflows could be designed, each configured to generate output with a particular form.

Document Generation and Print Options

In addition to providing a data file, and optionally specifying a form file, a number of other processing options are available for customizing how the generated PDF, or printed output, will be produced.

Generic options include: specifying whether Content Root and Locale; PDF specific options include the desired PDF version and tagged or linearized PDF; and, print specific options such as adjusting the pagination, page count etc.

Printing is achieved in AEM forms using SendToPrinter service; it supports shared printer, CUPS, LPD, CIFS, and direct printer IP printing methods.

For more information on the generic, PDF, and print options, consult the AEM forms documentation for a discussion of the `PrintedOutputOptions`, `PDFFormRenderOptions`, and `SendToPrinter Service` interfaces respectively.

Testing and Previewing

Testing your form, and its data bindings, from within the Designer software saves both time and paper, by avoiding the need to deploy the forms to the server and generate a printed test output.

Output Designer provides the ability to test both static and dynamic forms with its Test Presentment feature. Forms may either be tested with an existing data file, or Output Designer can create a sample data file suitable for testing the form. Forms can be viewed as a PDF using Reader or printed to a target printer.

AEM forms Designer provides an equivalent capability to test and preview forms with sample data. The resulting form, populated with data, is viewed as a PDF preview directly within the Designer software. From within the Designer's Form Properties dialog, the type of preview may be chosen as either a static interactive form, a dynamic interactive form, or a non-interactive printable form. The form may be previewed with an existing XML data file, or sample data may be generated by clicking the Generate Preview Data button from within the Form Properties dialog.

Another convenient way of performing a print test is available from the Print dialog within AEM forms Designer. In addition to the common options provided by the Print dialog, AEM forms Designer extends the dialog with settings to specify sample data, whether the form should be printed in a simplex or duplex mode, and the target print device configuration. In this way the form can be printed directly to a target printer to see a printed result.

Device Profiles

Forms created with Output Designer are designed specifically for one or more specific output formats or devices. However, forms created with AEM forms Designer are not constrained, or targeted, for a particular format or device; they are generic forms that AEM forms Output Service uses to produce output to a particular format or device at run-time, rather than making this decision at design-time.

Output Designer provides a mechanism for you to select one or more target formats and printers, by selecting from the available presentment targets. Depending on which presentment targets you select, a range of page sizes and fonts become available based upon the paper and font support of the devices. When creating a new form in Output Designer, your initial workspace, page size and orientation, and available fonts are all determined from the particular presentment target that has been nominated as the default presentment target. Before a form can successfully generate output to a presentment target, the form must be compiled after selection of the presentment target with Output Designer.

Advanced users of Print Agent and Output Designer may be aware that each supported presentment target is actually described by the contents of a text file, located within the product installation directory, with an `.ics` file extension. Hence, these files are commonly called **ICS files**. Because an ICS file describes the supported features and capabilities of a Central presentment target, the Central documentation references ICS files as the means for an advanced user to create or customize an ICS file in order to more accurately support their particular printer. For example, while Central provides built-in support for finishing options, such as stapling, on a number of supported printers, you may have a requirement to print forms on a compatible PCL or PostScript printer that supports stapling via a vendor-specific printer command. By customizing an ICS file to include the printer's specific stapling command, or creating a new ICS file based upon an existing compatible ICS file, Print Agent will be able to utilize the printer's stapling feature.

AEM forms Designer does not require that a form be compiled, nor does it require the selection of specific output formats or printers during the process of designing a form. When designing a form with AEM forms Designer, the page size and font capabilities of your target printer remain an important consideration. AEM forms Designer provides a wide variety of pre-defined page sizes, and permits the creation of custom page sizes.

The set of fonts available within the AEM forms Designer workspace is primarily determined from the fonts present on the system. In order to accommodate unique font requirements, where the font may not be present on the system, AEM forms provides font mapping capabilities for the server-side runtime components of AEM forms Output Service, and the AEM forms Designer environment.

The supported features and capabilities of printers supported by AEM forms Output Service are described by **XDC files**, which are conceptually similar to Output Designer's ICS files. In the same way that ICS files can be modified to accommodate the unique features of a particular printer, XDC files may also be customized, or new XDC files may be created based upon an existing XDC file.

AEM forms Designer also utilizes its own XDC file, `Designer.xdc`, to determine the range of page sizes, additional fonts, and other features, that will be exposed in the AEM forms Designer workspace.

AEM forms Output Service provides an additional XDC file, `Designer.xdc.label`, appropriately configured for designing printer labels, with page sizes and fonts appropriate for use with supported label printers. By copying the `Designer.xdc.label` file to a file named `Designer.xdc` (backup the original `Designer.xdc` first), and after restarting AEM forms Designer, these label-specific features will be exposed in the AEM forms Designer workspace.

Font Handling

Font Availability and Mapping

The set of fonts available within Output Designer is determined by the fonts available within selected presentment target. For instance, if the presentment target selected is the Generic Microsoft Windows Driver, all of the fonts available on the system are available. On, the other hand, if the presentment target is the Adobe Portable Document Format, then the available fonts correspond to the set of standard PDF fonts. Output Designer also provides mechanisms to further customize the set of available fonts associated with a presentment target. Font mapping can also occur when the final document is generated by Print Agent.

The set of fonts available within the AEM forms Designer workspace is primarily determined from the fonts present on the system, plus any fonts enumerated in the `Designer.xdc` file (as described in the previous section the section called "*Device Profiles*"). When ensuring that fonts are present on your systems, consider that fonts must be available both on the systems running AEM forms Designer, as well as the servers running the AEM forms server components.

In order to accommodate the unique font requirements of printers, AEM forms Designer and AEM forms Output Service provide a robust font mapping capability. AEM forms Designer, within the product installation directory, has a `Designer.xci` configuration file that contains the default set of font mappings. This configuration file can be extended to accommodate additional font mappings. However, this configuration file only impacts the AEM forms Designer workspace. Font mappings intended to be in effect at the time of processing a form in concert with a particular XDC device profile must also be present in the target XDC file.

The following is an excerpt from the `Designer.xci`, showing the XML markup describing a number of font mapping statements that will map requests for a variety of Helvetica fonts to similar Arial fonts:

Designer.XCI font mapping

```
<equate from="Helvetica Black_*_*" to="Arial Black_*_*" force="0"/>
<equate from="HelveticaBlack_*_*" to="Arial Black_*_*" force="0"/>
<equate from="Helvetica-Black_*_*" to="Arial Black_*_*" force="0"/>
<equate from="Helvetica_*_*" to="Arial_*_*" force="0"/>
<equate from="Helv_*_*" to="Arial_*_*" force="0"/>
```

The font mapping capabilities provide for very fine-grained control over mapping. Whole typefaces can be mapped, or an individual typeface with a particular weight and posture can be mapped. The force attribute denotes whether a font should be always mapped, or only when the requested font is not available.
The order of the equate font mapping statements is important, with the statements evaluated in order until a matching statement is encountered.

Paper Handling

Duplexing

Output Designer and Central provided a number of different ways of indicating whether a form should be printed simplex (one-sided) or duplex (double-sided), and whether pages should be duplexed along the left/long edge or top/short edge. These settings can be specified as a property of the whole form, from within a field-nominated data stream, or as an option on the job definition or Print Agent command-line.

AEM forms Designer exposes control over duplex printing in two ways, depending on whether the form is intended to be generated to a PDF and subsequently printed from Adobe Reader or Acrobat, or printed directly to a PCL or PostScript device.

When the form is intended to generate a PDF, settings related to how the PDF should be printed can be configured from the Form Properties dialog within AEM forms Designer. These settings include the number of copies to print, and duplexing. Subsequent printing of a PDF document with Adobe Reader or Acrobat will respect these settings.

Duplexing can also be specified via the `pagination` property on the AEM forms Output service operations, and also available from the Java API and web- service interface. For more information on the generic, PDF, and print options, consult the AEM forms documentation for a discussion of the `PrintedOutputOptions`, `PDFOutputOptions`, and `SendToPrinterService` interfaces respectively.

In addition to basic selection of simplex or duplex printing, there are additional capabilities available in AEM forms Designer to design forms that adjust according to simplex and duplex printing scenarios.

Master pages can be assigned to the odd numbered (front side) or even numbered (back side) printed surfaces. In this way, slightly different master pages can be designed for the front and back, and AEM forms will automatically select the appropriate master page depending on whether it is currently printing on the front or back of the page. One common use case for this capability is to create similar master pages that place the running page count on either the left or right side of the page, and assigning the master pages as odd or even to ensure that the page count is always on the inside or outside of a duplex printed document.

On a finer-grained level AEM forms Designer provides a presence property on form objects that commonly is used to indicate that an object should be visible or hidden from a PDF screen display of the document or when printed. Form objects can also be configured with a presence setting that indicates the object should only appear when the document is printed simplex, or duplex.

7. Web Output Pak

The preceding chapters have primarily focused on Central document generation. However, the Web Output Pak, as part of the Central solution, provides a similar set of document generation capabilities. In fact, Web Output Pak uses the document generation and output software component from Central, and uses forms created with Output Designer.

Web Output Pak processes data received from a web browser, or in XML format, and data is manipulated via a programmatic interface similar to an XML DOM. Concepts and experience gained from working with XML and DOM-style interfaces in Web Output Pak is beneficial when moving to the standards-based XML tools and interfaces of AEM forms Output Service.

Because Web Output Pak is a subset of the overall Central product family, the other sections of this document are also relevant in the context of many Web Output Pak solutions, and this section simply addresses several topics specific to Web Output Pak.

XPR and Transaction Processing

Web Output Pak applications are developed by coordinating the data handling, invocation of built-in and custom software agents, and output generation, by a markup language known as **XML Processing Rules** (**XPR**) and a rule processing engine known as the **Transaction Processor**. Because XPR markup is effectively a special-purpose programming language, there is no migration or conversion mechanism for XPR files to an equivalent mechanism in AEM forms Output Service

In place of Transaction Processor and XPR, AEM forms provides the AEM **Workflows** component and the ability to assemble Workflow Steps.

For more information, consult the product documentation on AEM Workflows.

Agents

Web Output Pak provides a number of built-in software agents, including the HTML Agent and PDF Agent that generate HTML and PDF output respectively.

The HTML Agent accepts data and populates regions of an HTML template file that contains Web Output Pak specific markup, generating a resulting HTML file that is a combination of the original HTML template and the supplied data. AEM forms Output Service does not provide a migration or conversion mechanism for these HTML template files. However, AEM forms Mobile Forms does have the capability to generate HTML output from forms created with AEM forms Designer.

The functionality of PDF Agent is equivalent to the PDF output capability provided by Central Print Agent, and generating PDF output is a core capability of AEM forms Output Service.

8. Hosting Environment

The Central product is a native application designed to run on a number of supported platforms (for more detail see table at Chapter 2, Comparison Summary. The hosting environment for Central is simply a supported operating system.

AEM forms is a module installed as a package in AEM (Adobe Experience Manager). AEM is a Java application can run as a standalone executable JAR file (just double click to start it) which includes an embedded application server, content repository, web application framework and full content management functionality in one package. AEM can also run as a war within a supported J2EE (Java 2 Platform, Enterprise Edition) application server environment, installed on a supported platform.

OSGi Application Framework

AEM is built within an OSGi application framework. OSGi is a dynamic module system for Java that provides a framework within which small, reusable standardized components can be composed into an application and deployed. These components, called bundles in OSGi parlance, can contain compiled Java code, scripts, and content to be loaded in the repository or configuration information. Bundles are loaded and deployed dynamically at runtime during normal operation of the application, enabling, for example, runtime upgrades to AEM without stopping the server

Apache Sling

AEM incorporates the Sling Web Application Framework that simplifies the writing of RESTful, content-oriented web applications. Sling uses a JCR repository, such as Apache Jackrabbit, or in the case of AEM, the CRX Content Repository, as its data store. Sling has been contributed to the Apache Software Foundation - further information can be found at Apache.

JCR Content Repository

AEM includes a Java Content Repository (JCR), a type of hierarchical database designed specifically for unstructured and semi-structured data. This repository is a data storage system specifically designed for content-centric applications. AEM uses this content repository to store all its web content, digital assets, scripts, Java libraries, configuration information and other data. AEM supports multiple deployment models starting from a simple TarMK instance to highly scalable MongoDB clusters.

It is also important to recognize that AEM Forms is designed as a server-based solution, and while installation on a workstation is useful for application development purposes, for production use AEM Forms should be installed on an appropriately configured server. The practice of installing many instances of Central across a number of workstations as a means of distributing processing and achieving some manner of scalability is not practical with AEM Forms.

- Deploying and maintaining AEM
- AEM Forms Architecture and Deployments
- Installing and configuring AEM forms

9. Field-Nominated Commands

The intent of this chapter is to review a number of the commands that comprise the field-nominated format used by Central agents, especially Print Agent, and provide a brief discussion of how the command relates to an equivalent capability in AEM forms Output Service. Not all commands supported by Central or Print Agent are included.

In most cases, the commands will not relate directly to XML markup that can be interpreted by AEM forms Output Service. This is because, as described in the section called "*Data Formats*", field-nominated data is both a command language and a data format, whereas XML data is strictly a data format.
Nonetheless, given that so much of the Central and Print Agent functionality is exposed through field-nominated commands, it is valuable to present them here as a way of referring to Output Service functionality.

^continue

See the section called "^record".

^copies

This field-nominated command, when it appears at the beginning of a field-nominated data file, sets the number of printed copies that shall be produced by Print Agent.

In AEM forms Output Service, the number of copies is set as a copies property on the `PrintedOutputOptions` interface

^currency

This field-nominated command sets a number of localization properties associated with the formatting of monetary values, such as the currency symbol, decimal point, thousands separator, and more.

XFA forms contain the equivalent property values for one or more locales associated with the form. The appropriate locale information is automatically incorporated into the form definition by selecting one or more locales from within AEM forms Designer. In addition, custom locales can be created. This is discussed in the section called "*Locale Settings*".

^data

See the section called "^record".

^define

This field-nominated command is used to assign a value to a dictionary variable.

With the field-nominated format there is no other way, other than dictionary variables, to store information in the data in a manner where it can later be recalled from elsewhere in the data or preamble processing. However, XFA forms operate with all of the XML data loaded into a DOM, accessible to scripting and data bindings. Therefore the need for an additional mechanism for storing, and later recalling, data is not required with XFA forms.

In addition, it should be noted that XFA forms may be designed to include form variables. For more information see the section called “*Form Variables*”.

^duplex

This field-nominated command controls how the document will be printed in duplex mode.

A XFA form cannot be set to duplex from within the data. Duplexing may be controlled from within AEM forms Designer as one of the form properties, or determined by specifying an appropriate `pagination` property passed to the AEM forms Output Service API. For more information see the sections called “*Duplexing*” and “*Device Profiles*”.

^eject

This field-nominated command causes Print Agent to indicate a page break in the generated output.

Within XFA forms, all control over pagination is specified within the form itself, using AEM forms Designer.

^field

This is the most commonly used field-nominated command. It directs the following data into a particular field indicated by name.

The equivalent mechanism in XML data is the markup tags themselves, where the tag names can optionally indicate the field name to which the enclosed data belongs. For a discussion of this, see the section called “*Data Formats*”.

^file

This field-nominated command causes a referenced data file to be included, at the location of the command, in the referencing data file.

One way to achieve this in XML is by using a markup feature related to XML known as **XInclude**. However, AEM forms Output Service does not currently support XInclude.

^form

This field-nominated command causes Print Agent to switch processing to another form file.

While XFA forms do not provide an equivalent capability of switching forms from within data, there is a capability to reference external fragments from within a form, and a feature to   automatically select an appropriate form based on the data known as search rules. For more information on search rules, see the section called “*Identifying the Form*”. For more information on fragments see the section called “*Fragments*”.

^global

This field-nominated command defines a global data value that will automatically populate fields with the same name as the global value.

XFA forms do not permit the data to indicate that values should be globally applied to form fields that share a common name. Global fields can be designated within AEM forms Designer, and there is a convention for defining a data value that can be available across multiple records of data. These capabilities are described in the section called "*Global Fields and Global Data*".

^graph

This field-nominated command specifies that a referenced image file, rather than a text value, shall be placed on the form at the location of a particular field.

XFA forms provide a specific type of field designed to enclose an image, known as image fields. Such fields can be populated with base-64 encoded data representing the image data, and additional properties are provided by AEM forms Designer to control how the image should be adjusted within the dimensions of the field. Alternatively, the HTML `img` element can be used to insert a referenced image file into a rich-text field. For more information, see the "Image Data and Rich-Text Reference" of the XFA Specification. The rich-text format is also briefly discussed in the section called "*Formatting Rich-Text Data*" of this document.

^group

This field-nominated command is intended to imply structure within a field-nominated format that would be otherwise unstructured. It also causes preamble processing to be executed associated with the referenced group, and is often used as a mechanism to instantiate a subform, transition to another page, or switch to another form entirely.

XFA forms rely upon the intrinsic capabilities of XML to describe structured data, and utilize this structure information when binding the data to a form. In this way, structure can cause data to populate specific subforms, on specific pages. In other words, the philosophy of utilizing XML data with XFA forms is to appropriately structure your data, appropriately structure your form into subforms and subform sets in AEM forms Designer, and express the relationship between the two structures by adding data bindings to the form.

^inlinegraphbegin, ^inlinegraphend

These field-nominated commands provide a mechanism to embed image data directly within a data file. See the section called "^graph" for more information.

^key

See the section called "^record".

^macro

This field-nominated command causes a previously downloaded macro, a printer-resident cached representation of a subform, to be immediately executed and included in the output.

This command provides a means to indirectly insert PCL or PostScript commands into the output stream. There is no equivalent capability in XFA forms.

^page

This field-nominated command causes Print Agent to perform a page break, and continue processing on a specific page of the current form.

Within XFA forms, the decision of when to perform a page break, and on which page to resume processing, is strictly determined by the pagination rules designed into the form with AEM forms Designer.

^passthru

This field-nominated command is a very low-level command that sends a printer-specific byte stream into the generated output, and subsequently to the print device. There is no equivalent capability in XFA forms. Some customization of AEM forms Output Service printer interface is possible by modifying device profiles. See the section called "*Device Profiles*".

^popsymbolset

See the section called "^symbolset".

^pushsymbolset

See the section called "^symbolset".

^record

This command is used to create a fixed-format record definition where the data is packed into rows and columns of a textual data file. AEM forms Output Service does not support fixed-record format data.

^symbolset

This field-nominated command informs Print Agent that subsequent data is encoded according to a specific character encoding. During the processing of data, Print Agent can be instructed to remember and recall specific character encodings with the `^pushsymbolset` and `^popsymbolset` commands respectively.

AEM forms Output Service processes XML data, and the XML specification requires that all of the character content of an XML document be expressed in a single encoding of Unicode characters. For more information see the section called "Unicode".

^shell

This field-nominated command provides a mechanism to execute an external software program from a specific point in the processing of the data file. AEM forms Output Service does not provide an equivalent capability.

^subform

This field-nominated command causes Print Agent to switch the currently active subform to a specific subform within the current form, or a subform within another form.

XFA forms to not provide a mechanism to call a specific subform from within the data. Instead, XFA forms rely upon the structure within XML data, and utilize this structure information when binding the data to a form. In this way, structure can cause data to populate specific subforms, on specific pages. In other words, the philosophy of utilizing XML data with XFA forms is to appropriately structure your data, appropriately structure your form into subforms and subform sets in AEM forms Designer, and express the relationship between the two structures by adding data bindings to the form.

^tab

This field-nominated command defines, or resets, tab stop positions. Tab stops can be set for the duration of applying data to the current field, or tab stops may be set that act as the default tab stops for the duration of processing the entire job.

AEM forms Output Service recognizes a subset of HTML markup to express text formatting within data, and within the form itself. Additional AEM forms-specific extensions to HTML markup are supported to specify tab stops. For more information, see the "Rich-Text Reference" portion of the XFA Specification.

^trayin, ^trayout

These field-nominated commands select a specified input tray, or output tray, respectively.

There is no capability in AEM forms Output Service to indicate tray selections from within the XML data. Each page of a XFA form may be configured to select a particular paper size, and tray selection is determined based on tray information associated with the paper size as defined by the device profile.

^$page

This field-nominated command causes the current page number to be set according to a supplied value.

Within XFA forms, page numbering is strictly a property of the form and of master pages in particular. For more information see the sections called "*Page Counts*", and "*Master Pages*".