Adobe® Digital Enterprise Platform Extension – Production Print

Version 10.0

User Guide

Rev A

POWERED BY OPENTEXT

Adobe® Digital Enterprise Platform Extension – Production Print User Guide
Rev A

Adobe® Digital Enterprise Platform Extension - Production Print 10.0

Contact information for Adobe Systems Incorporated

For patch updates, technical notes, and additional information about this product see

http://www.adobe.com/support/livecycle/

For other general questions see

www.adobe.com/aboutadobe/contact.html

Developer information

At the Adobe Developer Connection ADEP website,

http://www.adobe.com/devnet/enterprise-platform.html,

you can get the latest developer information and extend your knowledge with articles, tutorials, code samples, downloads, and sample applications.

For information about developer resources that are available, see

www.adobe.com/enterprise/developer/

Contents

About Production Print

Production Print is an extension of Adobe® Digital Enterprise Platform (ADEP). In this document, the ADEP Production Print Extension is referred to as Production Print.

Production Print enables organizations to address production-level data center output requirements by dynamically generating personalized documents for output in various formats (for example, AFP, PostScript®, PDF or PCL), using XML data from core applications. Production Print extends the reach of ADEP and ADEP Designer, allowing customers to standardize on a single design environment for all business critical communications.

Production Print provides production printing capabilities for the ADEP product suite. The solution is an OpenText – Adobe integrated solution where ADEP Designer is integrated into the OpenText StreamServe Persuasion environment.

This document provides information specific to Production Print. For general StreamServe Persuasion information, see the standard StreamServe Persuasion documentation. The reader has a basic understanding of StreamServe Persuasion and has ideally attended the StreamServe Persuasion Essentials course.

StreamServe Persuasion

Production Print is based on StreamServe Persuasion SP5 – Enhancement Pack 1. StreamServe Persuasion consists of:

- StreamServe Component Framework, which includes the underlying software to run StreamServe applications.
- StreamServer, the software to run StreamServer applications.
- Design Center, the main design tool in StreamServe Persuasion. In Design Center you can:
  - Create your StreamServe Projects.
  - Configure how to connect to the source application.
  - Identify and extract input data.
  - Configure how to deliver output to the output devices.

User documentation

Standard StreamServe Persuasion documentation is used along with the Production Print specific documentation.

The following documents are specific to Production Print:

- *Adobe Digital Enterprise Platform, Extension – Production Print, Release Notes* – describes new and changed features.
- *Adobe Digital Enterprise Platform, Extension – Production Print, User Guide* – (this document) describes the Production Print functionality.

- *Adobe Digital Enterprise Platform, Extension – Production Print, Installation Guide* – describes how to install, upgrade, and verify the Production Print software. It also lists supported software and platforms.

Introduction

ADEP Designer is the Adobe forms design tool, integrated into the StreamServe Persuasion design environment and used as a StreamServe Process tool. Output is produced in various formats, visually identical to the corresponding output from ADEP Document Services Output.

StreamServe Design Center can use ADEP Designer from either a stand-alone installation or from a version bundled with ADEP Document Services Workbench.

You can merge an existing form template with an XML instance document, or you can create a new form template (with bindings to non-XML data formats) in StreamServe Design Center.

The StreamServe Process tool for ADEP Designer can be used to:

- Import an existing ADEP Designer form template.
- Create a new ADEP Designer form template.
- Edit a form template using ADEP Designer.
- Export preview files and schemas for non-XML data formats for use in a stand-alone ADEP Designer.

Running without ADEP Designer

If you use an existing form template, you can run the StreamServe Process tool without ADEP Designer installed. However, you will not be able to edit the form template in the StreamServe Persuasion design environment.

In this chapter

Templates

Production Print processes XDP templates generated by ADEP Designer. It does not process ADEP Designer PDF files that include imported PDF artwork or Adobe PDF AcroForm format.

Templates can be loaded from a resource set in the Design Center Project or from the ADEP Document Services Repository.

From Design Center resource set

Templates loaded from a resource set in the Design Center Project are statically associated with the Process. These templates are loaded at start-up.

From Document Services Repository

Templates loaded from the Document Services Repository can be treated in different ways:

- Loaded during design time and stored in Design Center resource set. Design Center connects to the Document Services Repository and loads the templates into the Design Center resource set. You can update them by loading from the Document Services Repository.
- Loaded dynamically during runtime by connecting to the Document Services Repository using StreamServe variables or SOM expressions. These templates can be loaded once for each job or once for each processed record.

For information on how to access and load templates dynamically, see *Using templates* on page 25.

Performance considerations

Normally, performance slows down considerably when loading templates dynamically. The performance is highly dependent on the file system, the file I/O and network capacity. To get maximum performance, make sure that the templates are accessible to the Process in the fastest possible way.

To improve performance when using dynamic templates, you can enable template caching. See *Template caching* on page 12.

Dependencies – Fragments and images

XDP files with dependencies (external references to fragments and images) are supported. A referenced file is not embedded in the main XDP. When importing a template with dependencies, the Process tool resolves the references and imports the needed resources.

StreamServer can process dependencies from the local file system, from the local network, from HTTP resources, from FTP resources, and from the Document Services Repository.

When importing a LiveCycle Archive file (LCA), the main XDP and all its dependencies are included in the imported LCA package.

Templates and related files can be imported from the Document Services Repository to a resource set in the Design Center Project. See *Accessing Document Services Repositories* on page 34.

Dynamically loaded templates can include dependencies to resources residing in the Document Services Repository. To enable this, you must select **Enable referenced resources** in the Select Template dialog box. See *Loading directly from Document Services Repository* on page 29.

Note: Updating the imported resources on their source location will not automatically update the dependencies imported to the Process tool. If dependencies are changed on the source location, you have to update them in the Process tool.

Post-processing

Post-processing in Production Print produces output in the same way as the StreamServe PageOUT tool.

When using an existing form template with an XML data set with several records, post-processing does not treat each record as a separate Message. Instead, records are treated as a batch within a Process.

In this chapter

Running scripts before and after each record

You can run Before and After Process scripts, before and after each record. This is enabled by an option in the Settings dialog box. See *Settings dialog box* on page 86.

The very first Before Process script and the very last After Process script, will run in both pre-process and process phase. The other Before and After Process scripts will run only in the process phase. It is important to consider this when configuring the scripts.

See *Using Before and After Process scripts* on page 106.

Grouping into logical documents – Document trigger

Normally, in Production Print, each record is automatically mapped to one document.

To be able to split and group the output from the Processes connected to the output connector, you can use the Document trigger. For example, this is useful if all documents with the same customer number in the input job should be included in the same document. the Document trigger is defined in the Runtime Output Connector Settings, see *StreamServer Persuasion SP5 Design Center* documentation.

You can use Document trigger for output modes Document and Job.

Using the automatic document trigger

To keep the behavior with automatic mapping of each record to one document, there is a setting **Automatic Doc Trigger**. When set it disables the Document Trigger variable.

This setting is by default selected for Projects upgraded from releases previous to LiveCycle Production Print ES2.

See *Runtime Process Settings dialog box – General tab* on page 97.

To set the automatic document trigger

1. In the Runtime view, right-click the ADEP Designer Process and select **Settings**. The Runtime Process Settings dialog opens.
2. Select the **General** tab.
3. Select the **Automatic Doc Trigger** option. The **Document trigger variable** (specified in the Runtime Connector Settings dialog, Document Trigger tab) is now disabled and each record will be automatically mapped to one document. There will be no grouping of output into logical documents.

Performance considerations

In this chapter

Font data caching

To improve performance you can use the `preloadmorefontdata` startup argument that turns on caching of additional font data at startup. This results in faster runtime execution, but slower startup time and increased memory consumption.

Default is no caching of additional font data.

See *StreamServe Persuasion SP5 Startup arguments* documentation.

Template caching

Loading and unloading form templates can affect the performance. When the same template is used several times, as is the case when using dynamic templates, performance can be significantly improved by caching.

Caching of templates is enabled/disabled in the Settings dialog box. See *Settings dialog box* on page 86.

If the original form template has been modified, it will be re-loaded automatically to the cache. The timestamps of the original template file and the cached file are used to determine if the original file has been changed. This only applies to the main template file. Modified fragments are not re-loaded.

You can set the size of the template cache in StreamServe Control Center and via the command line. The cache size is set in KB, and it will override the default cache size value.

Default cache size is 10240 KB.

Control Center

The option **Cache size** is set in the Properties view.

Command line

```
-maxcachesize <value>
```

Supported output formats

The XFA processor supports the following output formats and drivers.

Output format	**Production Print driver**
AFP – Advanced Function Presentation	AFP
IJPDS – Ink Jet Printer Data Stream	IJPDS
PDF – Portable Document Format	PDF
Tagged PDF	PDF
PDF/A 1a	PDF
PDF/A 1b	PDF
PCL – Printer Control Language	PCL5
P.S. – Postscript	Postscript
TIFF – Tagged Image File Format	TIFF
ZPL II – Zebra Programming Language	Zebra ZPLII
FP/DP – Intermec Fingerprint/ Direct Protocol	Intermec FP/DP
IPL – Intermec Printer Language	Intermec IPL
PGL/IGP – Intelligent Graphics Printing/ Printronix Graphics Language	Printronix PGL/IGP
Toshiba TEC	TEC
Windows Print API	Windows Driver (StreamServe 3.x)

Other StreamServe Persuasion output formats are not tested, but may work with limitations to font and object rendering.

See the standard StreamServe Persuasion documentation for information on output formats supported by StreamServe Persuasion SP5.

Windows driver

You can create a Production Print driver configuration file (DRS) for the Production Print Windows driver, for use with specific third-party printer drivers. See *StreamServe Persuasion SP5 Device Driver Tools* documentation.

Requirements for exact rendering of bulleted lists

The following fonts are required to provide the exact same line spacing in the output as in the preview in ADEP Designer (WYSIWYG):

- Symbol, Type 1 version – Required for black dot (0xB7).
- Adobe Pi Std – Required for black square (U+25A0).
- Courier Std – Required for white circle (U+25CB).

The fonts must be available in the driver configuration file (DRS) and to the StreamServer application that formats and distributes the output document.

For information on how to update DRS files, see the *StreamServe Persuasion SP5 Device Driver Tools* documentation.

For information about how to make fonts available to a StreamServer application, see *Loading fonts* on page 30.

Hyphenation

Production Print supports hyphenation much like ADEP Document Services Output and ADEP Document Services Forms.

For known hyphenation issues, see *Adobe Digital Enterprise Platform, Extension – Production Print, Release Notes.*

Using an existing form template

This chapter describes how to enable an existing form template for production printing. The XML instance document is merged with a form template.

How it works in StreamServer

1. The XMLIN Event detects the XML instance document and triggers the Process.
2. The Process merges the content of the input XML instance document with the form template and produces structured output data.
3. The produced output data is sent to driver and post-processing for formatting.
4. The formatted output is sent to its destination.

In this chapter

Configuring the Project

Prerequisites

- An XML instance document as input data.
- A form template, containing data bindings to the XML instance document.
- A StreamServe Project, which is open in Design Center.

To add the form template as a resource

1. In the Design Center Project browser, double-click the resource set. The resource set view opens.
2. Right-click the resource set, select **Import** and browse to and select the file to import as a resource. The resource is created and added to the resource set.

To create the Message

1. Select **File** > **New** > **Message**. An empty Message is created.
2. Rename the Message.

To create the Event

1. Right-click anywhere in the Message view and select **Add Event** > **XMLIN**. A new Event is added to the Message view.
2. Rename the Event.
3. Right-click the Event and select **Open**. The Event tool is opened.
4. Open the XML instance document as a sample by selecting **File** > **Open Sample.** The Select Resource dialog box is opened.
5. Browse to the file and select the file. The file is displayed in the XMLIN Sample view.
6. Use the Pattern tool to create a pattern that will trigger the Event. See *StreamServe Persuasion SP5 XMLIN* documentation.
7. Save and exit the Event tool.

To create the Process

1. Right-click anywhere in the Message view and select **Add Process** > **ADEP Designer**. A new Process is added to the Message view.
2. Rename the Process.
3. Right-click the Process and select **Open**. The Process tool is opened with the Settings dialog box displayed. See *Settings dialog box* on page 86.
4. Click **Load ADEP Designer GUI at startup** and **OK**. Note that this is optional, you may not have the ADEP Designer installed.

   ADEP Designer is launched with an empty drawing area. In the Data view, data connection is empty because there is no Message connected.

5. Import the form template by selecting **File** > **Open/Select Template**. The Select Template dialog box opens.
6. Select the **From Design Center resource set** check-box, browse for and select the form template file. ADEP Designer is launched with the form template displayed in the drawing area.
7. Save and exit the Process tool.

To finalize the Project

1. Configure the Platform.
2. Configure the Runtime.

   Note: You can use the **Ignore remaining data** option in the Runtime Event settings to improve performance. This option is used to ignore remaining data when the trigger pattern is found.

3. Export the Project.

See *StreamServe Persuasion SP5 Design Center* documentation.

The next step is to deploy an run the Project, see *Deploying and running* on page 20.

Deploying and running

To deploy the Project

Deploy a Project to a StreamServer application in Control Center:

1. Create the StreamServer application to deploy the Project to.
2. Deploy the Project to the StreamServer application.

See *StreamServe Persuasion SP5 Control Cente*r documentation.

To run the StreamServer application

Start the StreamServer application in Control Center by right-clicking the StreamServer application node and select **Start**.

You also stop and redeploy StreamServer applications in Control Center. See *StreamServe Persuasion SP5 Control Cente*r documentation.

Using a form template created from scratch

This chapter describes how to:

- Create a form template from scratch, using a StreamServe Message for field and block bindings.
- Enable the template for production printing.

How it works in StreamServer

1. The Event detects the input file, extracts the data to a Message and triggers the Process.
2. The Process merges the content of the Message with the form template and produces structured output data.
3. The produced output data is sent to a driver and post-processing.
4. The formatted output is sent to its destination.

In this chapter

- *Configuring the Project* on page 22.
- *Deploying and running the Project* on page 24.

Configuring the Project

Prerequisites

- A StreamServe Project, which is open in Design Center.
- Input file containing field-based input data.

To create the Message

1. Select **File** > **New** > **Message**. An empty Message is created.
2. Rename the Message.

To create the Event

1. Right-click anywhere in the Message view and select **Add Event** and Event type. For field-based input, Event type is StreamIN. A new Event is added to the Message view.
2. Rename the Event.
3. Right-click the Event and select **Open**. The Event tool is opened.
4. Configure the Event. See *StreamServe Persuasion SP5 Design Center* documentation.
5. Save and exit the Event tool.

To create the Process

1. Right-click anywhere in the Message view and select **Add Process > ADEP Designer**. A new Process is added to the Message view.
2. Rename the Process.
3. Right-click the Process and select **Open**. The Process tool is opened with the Settings dialog box displayed. See *Settings dialog box* on page 86.
4. In the Settings dialog box, select **Load ADEP Designer GUI at startup** and **Add the Message as a data connection in the Data View** options.
5. Click **OK**. The Settings dialog box is closed and ADEP Designer is launched with an empty drawing area and the content of the Message tree as a data connection.
6. Configure the Process and move fields from the Message to the drawing area using drag-and-drop. A form object with a binding to the field is created for each field.
7. Select **Save**. When saving for the first time, the Select resource for storing main XDP template dialog box is opened.
8. Select resource and click **OK**.

To finalize the Project

1. Configure the Platform.
2. Configure the Runtime.

3 Export the Project.

See *StreamServe Persuasion SP5 Design Center* documentation.

The next step is to deploy an run the Project, see *Deploying and running the Project* on page 24.

Deploying and running the Project

To deploy the Project

Deploy a Project to a StreamServer application in Control Center:

1. Create the StreamServer application to deploy the Project to.
2. Deploy the Project to the StreamServer application.

See *StreamServe Persuasion SP5 Control Cente*r documentation.

To run the StreamServer application

Start the StreamServer application in Control Center by right-clicking the StreamServer application node and select **Start**.

You also stop and redeploy StreamServer applications in Control Center. See *StreamServe Persuasion SP5 Control Center* documentation.

Using templates

Note: Loading templates dynamically can have a negative effect on performance. See *Performance considerations* on page 12.

References in dynamically loaded templates

Templates with dependencies (external references to fragments and images) are supported if the referenced files are accessible to the Process via:

- Static paths.
- Paths that are located relatively to the form template location in a file system.
- HTTP URL.

References in templates from Document Services Repository

Templates loaded dynamically from the Document Services Repository can contain references to other assets in the repository. For example, to fragments and images. To enable this, you must select **Enable referenced resources** in the Select Template dialog box. See *Select Template dialog box* on page 89.

Application versions

There might be multiple versions of a resource in the Document Services Repository bound to different ADEP application versions. Production Print will not automatically use the latest application version.

In this chapter

Using StreamServe variable to load template

You can use a StreamServe variable when the template path cannot be specified by a SOM expression, or when the path must be specified by scripting in StreamServe.

The variable is evaluated once for each job, directly after the Process is started. This means that the template is used for all records processed in the job.

The variable is a string and both file paths and URIs (file, HTTP, and repository URIs) are supported. Paths and URIs can be absolute or relative (to the StreamServer working directory).

For HTTP URIs, simple HTTP authentication can be used.

Example 1 *File path and URI examples*

- File path:
  `C:\templates\mytemplate.xdp`
  or
  `../../MyTemplate.xdp`
- File URI:
  `file:///D:/my%20templates/mytemplate.xdp`
- HTTP URI:
  `http://examplehost/mytemplate.xdp`
- Repository URI:
  `repository:///myfolder/myresource.xdp`

Example 2 *Template selection based on external data*

The template file used for a particular user category is stored on disk (or in a database).

A Before Process StreamServe script is used to read a value from the file, based on the user category, and assigns the template path to the variable. The variable is then used for template selection in the Process.

Absolute path example:

```
$template =”C:\templates\dynamic_invoice.xdp”;
```

Relative (from working directory) path example:

```
$template = “../data/XDP_template/ dynamic_invoice.xdp”;
```

Example 3 *Template selection based on metadata*

The template path is sent to StreamServe as an HTTP header value.

A script extracts the value from the HTTP header and assigns the value to a variable. The variable is then used for template selection in the Process.

To load a template during runtime using a StreamServe variable

1. In the Process Tool, select **File** > **Open/Select Template**. The *Select Template dialog box* opens.
2. Select **StreamServe variable** and enter the name of the variable pointing to the template.
3. If connecting to a Document Services Repository, enter the **Runtime repository connection** details.
4. If using HTTP URI, and simple HTTP authentication, select **Use Simple HTTP Authentication** and enter the logon credentials.
5. Click **OK**. The starting window for the Process tool opens.

Using SOM expression to load template

Using the SOM expression

The SOM expression is useful when the path to a template is accessible from the data DOM and when you need a different template for every record in a batch job.

The SOM expression can be evaluated once for each record or once for each page processed. This means that different templates can be used for each processed record or page.

File paths and URIs are supported in the same way as for StreamServe variable, see *Using StreamServe variable to load template* on page 26.

Example 4 *SOM expression*

The template path is located in an element in the input file.

A SOM Expression pointing to the element is used for template selection in the Process.

To load a template during runtime using a SOM expression

1. In the Process Tool, select **File** > **Open/Select Template**. The *Select Template dialog box* opens.
2. Select **SOM expression** and enter the SOM expression in the data DOM pointing to a template.
3. If connecting to a Document Services Repository, enter the **Runtime repository connection** details.
4. If using HTTP URI, and simple HTTP authentication, select **Use Simple HTTP Authentication** and enter the logon credentials.
5. Click **OK**. The starting window for the Process tool opens.

Loading directly from Document Services Repository

Templates loaded from the Document Services Repository are loaded statically at design time. StreamServer will load the templates from the repository at runtime. This means that the user can update a template and store a new version in the repository during runtime. The server will load the latest stored template.

To load template from the Document Services Repository

1. In the Process Tool, select **File** > **Open/Select Template**. The *Select Template dialog box* opens.
2. Enter the **Runtime repository connection** details for the Document Services Repository.

   Optionally, select **Enable referenced resources** to resolve references to resources in the Document Services Repository. For example, references to fragments and images.

   Note: The **Enable referenced resources** option may have negative impact on performance.

3. Select the **From ADEP Document Services Repository** option and browse to select a template from the repository. The Runtime repository connection details are used to connect to the repository.
4. Click **OK**. The starting window for the Process tool opens.

Loading fonts

For performance reasons, StreamServe loads all fonts during StreamServer startup.

When using statically loaded templates, the Design Center export package contains all referenced fonts.

When using dynamically loaded templates, you have to manually include the fonts to be used by the dynamic templates.

This means that you have to add the fonts to the resource set in a Project.

To add the fonts to the resource set in a Project

Manually import the fonts into the Project, as described in the *StreamServe Persuasion SP5 Design Center* documentation.

Hint – alternative procedure

Another way to manually include the fonts: import one or more XDP templates, containing all the fonts you expect to use as resources, and connect them one by one to a Process tool (ADEP Designer).

Save the Process after you have attached each XDP. This will automatically import all used fonts to the Project.

You can optionally remove the XDP resources when done. See the *StreamServe Persuasion SP5 Design Center* documentation.

Working with external templates

You can use an XML schema file, exported from a StreamServe Message, to edit and design a template in ADEP Designer stand-alone. The exported file uses the StreamServe Message as data connection in the Data view. The template can be re-imported into the StreamServe solution. This is useful, for example, if you outsource development and maintenance of templates.

You can create a preview of the XML schema with sample data. Sample data can be entered for each field in the StreamServe Event tool, for example PageIN or XMLIN.

This is particularly useful when the input data is a non-XML data format, such as ASCII text. Production Print transforms the input to XML internally, but if you want to do a preview in ADEP Designer, there is typically no XML file to use for the preview. You can use the **Export Preview XML** function to create one in this scenario.

Usage scenario

A scenario can be that you have outsourced the design of the template to an external designer. You create two exported files – a Message schema file and a preview file with sample data – and hand them over to the external designer. When the external designer is finished, you re-import the template into Design Center.

To export a StreamServe Message

1. In the Process tool, select **File** > **Export Message Schema** command. The Save As dialog box opens.
2. Browse to and select location and name for the schema (XSD) file.

To create a preview XML file with sample data

1. Open the Settings dialog box and check that the option **Add the Message as a data connection in the Data View** is selected.
2. Create a preview XML file with sample data, select **File** > **Export Preview XML**.
3. Browse to and select location and name for the schema (XSD) file.

To re-import an exported template into a Design Center resource set

1. Open the resource set view.
2. Select the **Import** command and browse to and select the template file to import. The Resource type settings dialog box opens.
3. Specify resource type **XDP Template** from the drop-down list and click **OK**. The resource is added to the resource set.

32 Working with external templates

Production Print and ADEP integration

StreamServe Design Center accessing Document Services Repository

StreamServe Design Center can be connected to a Document Services Repository. This makes it possible to use templates and other related resources with Production Print without having to import them via the file system. The templates and related files are imported from the Document Services Repository to a resource set in the Design Center Project.

ADEP invoking Production Print applications

ADEP can invoke StreamServer applications that are exposed through web services. These web services can be used to integrate StreamServer applications into ADEP processes when processing documents.

Production Print invoking ADEP processes

StreamServer can invoke ADEP processes that are deployed within ADEP and exposed through web services. These web services can be used to integrate ADEP processes into the StreamServer pipeline when processing documents.

In this chapter

Accessing Document Services Repositories

Templates and related files can be imported from the Document Services Repository to Design Center and stored as resource sets.

By accessing the Document Services Repository from Design Center you can:

- Navigate and browse the Document Services Repository. The browser shows information about:
  - If the resource already exists locally in the Design Center resource set.
  - If it has been updated in the repository and needs to be updated locally in the Design Center resource set.
  - If it has been updated locally in the Design Center resource set.
  - If it has been moved from the repository.

  See *Icons used when accessing Document Services Repository* on page 99.
- Import resources and their dependencies from the Document Services Repository.
- Update already imported resources from the Document Services Repository.

You cannot change any data in the Document Services Repository when accessing it from Design Center; you only have read access.

Note: You can define connections to several repositories, but you can only create and update resources from one repository at the time.

Connecting to the Document Services Repository

You can define and activate the connections to a Document Services Repository from Design Center. You can set one connection as active at the time. The connection will only be active during communication (during import and update).

Select **Tools** > **Select ADEP Document Services Repository connection**. The *Select Active ADEP Document Services Repository Connection dialog box* opens.

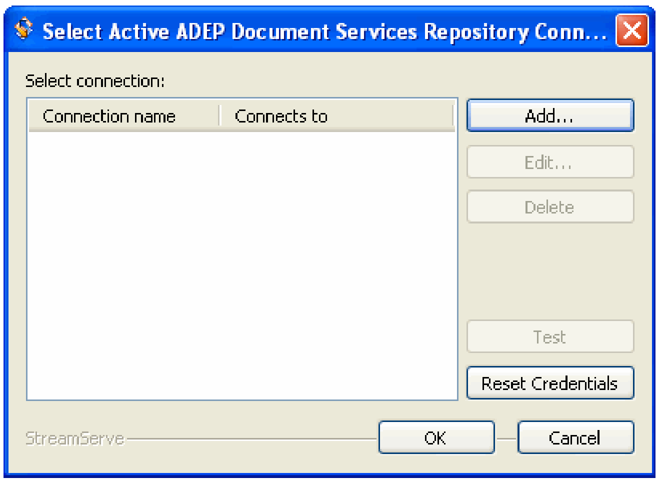


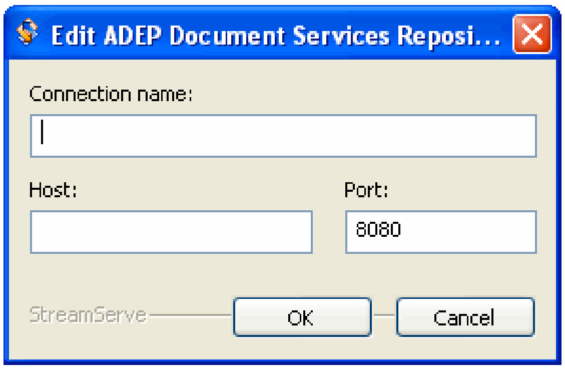
Figure 1 The Select Active ADEP Document Services Repository connection dialog box.

To activate a connection

Select the check box for the connection to be activated. The selected connection will be activated when needed (that is, during import and update from the repository).

To add or edit a connection

1. Click **Add** or **Edit**. The Edit ADEP Document Services Repository Connection dialog box opens.
2. Specify the settings.



Port: 8080

- **Connection name** – Choose an appropriate name for the connection.
- **Host** – Host name or IP address of the server were the repository is located.
- **Port** – The port used for communication with the host.

3. Click **OK**. The connection must exist and be available; when added the connection is accessed and identified. If it does not exist, you will get an error message.

To delete a connection

1. Select (highlight) the connection to delete.
2. Click **Delete**.

To test a connection

You can test if a connection to a Document Services Repository works.

1. Select (highlight) the connection to test.
2. Click **Test**. You will be prompted for logon credentials.
3. Enter your logon credentials and click **OK**.

Importing a resource from the Document Services Repository

When importing a resource from the Document Services Repository, a local copy of the resource is created in the Design Center resource set.

The resource will be added in a path and file structure which reflects the structure in the Document Services Repository.

Note: Do not change the structure in the Design Center resource set. References to fragments and images may be broken if their internal relative positions are changed.

You can choose to import a resource with or without dependencies.

To import a resource

1. Select **Resources** > **Import from ADEP Document Services Repository**. The Select resource dialog box opens.
2. Browse to and double-click the resource to import. The *Import ADEP Document Services Repository Resource dialog box* opens.

   The option **Always check out Head version of all Resources** is selected by default. This means that the latest version of the resource and its dependencies will be imported.
3. If you wish to import another version than head version, unselect the check-box and select version from the drop-down list.
4. Select resource and dependencies to import and click **OK**.

Updating resources from the Document Services Repository

To update a resource and its dependencies

When the original resource (in the Document Services Repository) has been changed, you can update the local copy in the Design Center resource set.

1. Right-click the resource in the resource set view and select **Update From Origin**. The *Update all ADEP Document Services Repository resources dialog box* opens, displaying the status of the local resource and its dependencies compared to the resource in the repository. See *Icons used when accessing Document Services Repository* on page 99.
2. Select resource and dependencies to import.
3. Click **OK**. The resource and the selected dependencies will be stored in the same path and file structure as in the Document Services Repository.

To update multiple resource

When there are a lot of resources changed in the Document Services Repository, you may wish to update multiple resources in one go.

1. Select **Resources** > **Update all ADEP Document Services Repository Resources**. The *Update all ADEP Document Services Repository resources dialog box* opens, displaying the imported resource and its status relative to the repository resources. See *Icons used when accessing Document Services Repository* on page 99.

   Note: The dependencies are not updated when using the **Update all ADEP Document Services Repository Resources** command.

2. Select which resources to update by clicking their check-boxes or **Select all.**
3. Click **OK**. The selected resources will be stored to the same path and file structure as in the Document Services Repository.

Invoking Production Print from ADEP

ADEP can invoke StreamServer applications that are exposed through web services. These web services can be used to integrate StreamServer applications into ADEP Document Services Processes when processing documents.

Input and output data

The StreamServer service is completely generic. Any type of data can be sent to StreamServer, for example an XML data file for merging with a form template.

The result from StreamServer is equally generic. It can be anything from a print file to a status message depending on the StreamServer configuration.

Document Service Component used for the integration

The Document Service Component (DSC) called Production Print DSC is developed for this purpose. This DSC can be used in any Document Services Process to pass data to and from StreamServer.

The Production Print DSC is packaged as a jar file (`lcppdsc.jar`) in the installation media. This DSC can be deployed in ADEP through Document Services Workbench.

StreamServer connectors used for the integration

The web services are exposed by StreamServer using Service Request input connectors. StreamServer receives the job from ADEP via the Service Request connector, and can return processed output to ADEP via any output connector.

Service Gateway

A Service Gateway must be running. The Service Gateway manages the web service calls between ADEP and StreamServer.

Sample Project

A Design Center Project, `sampleproject.dcpackage`, is provided on the installation media. This Project has a sample of this usage – it has the ADEP process side as well as the Production Print side. Studying and trying this sample Project is recommended. This sample Project is also recommended as a pattern for developing your own integrated processes. See *Sample Projects* on page 57 for more information on this sample Project.

Production Print DSC characteristics

The Document Services Workbench service has three functions:

- *Post* – Sends a job from ADEP to StreamServer. No status information is returned.
- *Run* – Sends a job from ADEP to StreamServer. Status information is returned when the output job from StreamServer is completed.
- *Generate* – Sends a job from ADEP to StreamServer for processing, and then receives the processed job and status information in a response from StreamServer.

Post

Use this function if you only want ADEP to send a job to StreamServer for further processing, and if no status information is required after the output job from StreamServer is completed.

Parameters

The parameters listed below apply to this function.

Parameter	Type and sub-type	Description
Remote Endpoint	**Type**: String	The Service Gateway address. For example: `http://localhost:2718`
Remote Service Name	**Type**: String	The name (case sensitive) of the StreamServer service to invoke. Must be exactly the same as `Service Name` on the Service Request input connector used by StreamServer to retrieve the job. See *Creating a Service Request input connector* on page 44.
Connection Timeout	**Type**: Integer	The maximum time (seconds) to wait for StreamServer to retrieve the job. A timeout set to <=0 means no timeout.
Input Data	**Type**: Document	The job (document, batch run, etc.) and content type of the job to be processed by StreamServer.
Template	**Type**: Document	XDP to be used by StreamServer when processing the job.
Enable referenced resources	**Type**: Boolean	Enables the service to resolve references to resources in the Document Services Repository and merge them into the XFA template.
Additional Input Parameters	**Type**: List **Sub-type**: Input Parameter	Variables to pass on to StreamServer. StreamServer must use the script function `GetConnectorValue` to access these variables.

Run

Use this function if you only want ADEP to send a job to StreamServer for further processing, and if you want StreamServer to return status information when the output job is completed.

Parameters

The parameters listed below apply to this function.

Parameter	Type and sub-type	Description
Remote Endpoint	**Type**: String	The Service Gateway address. For example: `http://localhost:2718`
Remote Service Name	**Type**: String	The name (case sensitive) of the StreamServer service to invoke. Must be exactly the same as `Service Name` on the Service Request input connector used by StreamServer to retrieve the job. See *Creating a Service Request input connector* on page 44.
Connection Timeout	**Type**: Integer	The maximum time (seconds) to wait for a response from StreamServer. If a timeout occurs when StreamServer processes the job, the job is not removed from the queue database. A timeout set to <=0 means no timeout.
Input Data	**Type**: Document	The job (document, batch run, etc.) and content type of the job to be processed by StreamServer.
Template	**Type**: Document	XDP to be used by StreamServer when processing the job.
Enable referenced resources	NA	Enables the service to resolve references to resources in the Document Services Repository and merge them into the XFA template.
Additional Input Parameters	**Type**: List **Sub-type**: Input Parameter	Variables to pass on to StreamServer. StreamServer must use the script function `GetConnectorValue` to access these variables.
Returned Status and Documents	**Type**: Result Status and Document(s)	Status and documents returned by StreamServer.
Sub-parameters to Returned Status and Documents		
statusCode	**Type**: Integer	Status code returned by StreamServer. 0: OK 1: Warning

Parameter	Type and sub-type	Description
statusMessage	**Type**: String	Additional status information. Contains detailed information related to the statusCode returned by StreamServer.

Generate

Use this function if you want ADEP to send a job to StreamServer for processing, and then retrieve the processed job in a response from StreamServer.

Parameters

The parameters listed below apply to this function.

Parameter	Type and sub-type	Description
Remote Endpoint	**Type**: String	The Service Gateway address. For example: `http://localhost:2718`
Remote Service Name	**Type**: String	The name (case sensitive) of the StreamServer service to invoke. Must be exactly the same as `Service Name` on the Service Request input connector used by StreamServer to retrieve the job. See *Creating a Service Request input connector* on page 44.
Connection Timeout	**Type**: Integer	The maximum time (seconds) to wait for a response from StreamServer. If a timeout occurs when StreamServer processes the job, the job is not removed from the queue database. A timeout set to <=0 means no timeout.
Input Data	**Type**: Document	The job (document, batch run, etc.) and content type of the job to be processed by StreamServer.
Template	**Type**: Document	XDP to be used by StreamServer when processing the job.
Enable referenced resources	NA	Enables the service to resolve references to resources in the Document Services Repository and merge them into the XFA template.
Additional Input Parameters	**Type**: List **Sub-type**: Input Parameter	Variables to pass on to StreamServer. StreamServer must use the script function `GetConnectorValue` to access these variables.
Returned Status and Documents	**Type**: Result Status and Document(s)	Status and documents returned by StreamServer.

Parameter	Type and sub-type	Description
Sub-parameters to Returned Status and Documents		
statusCode	**Type**: Integer	Status code returned by StreamServer. 0: OK 1: Warning
statusMessage	**Type**: String	Additional status information. Contains detailed information related to the statusCode returned by StreamServer.
documents	**Type**: List **Sub-type**: Document	The job, and content type of the job, returned by StreamServer.

Error handling

In case of errors, the exception `ProductionPrintException` will be thrown. The error codes are described in the table below.

Code	Description
`-1`	Service server error. This is an error from which the client cannot recover by simply retrying. For example, out of memory or out of disk space on the server.
`-2`	Client error. This is an error from which the client may recover. For example, invalid service name or too short time out.
`-3`	Invoke error. An invalid SOAP Envelope was sent to the web service.
`-4`	Remoting error. For example network failure, or invalid end point specified.
`-5`	Output data error. The client could not receive output data from the server.
`-6`	Addressing error. An invalid end point was specified for the web service.
`-7`	Local IO error. An IO error on the DSC side. For example, out of disk space on the DSC host.
`-8`	Generic DSC error. The error message in the exception contains more details.

StreamServer configuration

The StreamServer configuration includes a Service Request input connector and the appropriate Event, Process, output connector, and queues.

Service Request input connector

This connector exposes the web service to ADEP, and retrieves the job from ADEP.

Event and Process

The Event and Processes are configured according to standard Design Center procedures.

Output connector

In a scenario where StreamServer delivers the final output, the output connector is configured according to standard Design Center procedures.

In a scenario where ADEP delivers the final output, the output connector must also be configured to return the job in the web service response.

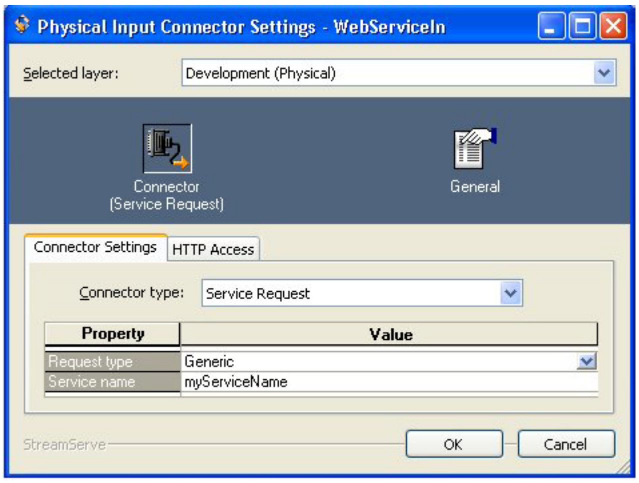
Queues

The input and output connector must be connected to queues.

Creating a Service Request input connector

You create a Service Request input connector the same way as you create other input connectors in Design Center.

Connector settings


Physical Input Connector Settings - WebServiceIn
Connector (Service Request)
General
Connector Settings
HTTP Access
StreamServe
OK
Cancel

Selected layer: Development (Physical)

Connector type: Service Request

Property	Value
Request type	Generic
Service name	myServiceName

Figure 2 Physical Input Connector Settings dialog box

Setting	Description
Request type	Select **Generic**.
Service name	The name of the web service to expose to ADEP.

Enabling service response

In a scenario where ADEP delivers the final output, StreamServer must be configured to return its output to ADEP in the web service response. This is done in the output connector configuration. Any type of output connector can be used, for example a Null connector.

Note: The output connector must be connected to an output queue.

To enable service response

1. In Design Center, activate the generic Platform layer.
2. Double-click the output connector. The Output Connector Settings dialog box opens.
3. Click the **General** icon, select **Include result in service response**, and click **OK**.

Retrieving variables

If variables are delivered in the service request from ADEP, StreamServer must use the script function `GetConnectorValue` to retrieve the variables. See the *StreamServe Persuasion SP5 Scripting Reference* documentation for more information on this script function.

Invoking Document Service Processes from Production Print

StreamServer can invoke Document Services Processes that are deployed within ADEP and exposed through web services. These web services can be used to integrate Document Services Processes into the StreamServer pipeline when processing documents.

Processes created and activated using Document Services Workbench can be invoked by sending the appropriate invocation request (SOAP request) to ADEP.

ADEP filter and ADEP output connector

There are two ways to invoke requests from StreamServer to ADEP:

- ADEP output connector – used when ADEP delivers the final output. See *ADEP output connector* on page 46.
- ADEP filter – used when StreamServer delivers the final output. See *ADEP filter* on page 51

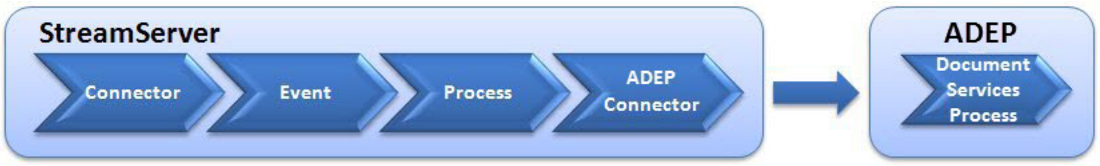
Sample Project

A Design Center Project, `sampleproject.dcpackage`, is provided on the installation media. This Project has a sample of this usage – it has the ADEP process side as well as the Production Print side. Studying and trying this sample Project is recommended. This sample Project is also recommended as a pattern for developing your own integrated processes. See *Sample Projects* on page 57 for more information on this sample Project.

ADEP output connector

The ADEP output connector is used when ADEP delivers the final output.

Example – ADEP output connector usage



1. StreamServer receives input via an input connector.
2. StreamServer uses the appropriate Event/Process configuration to create documents.
3. The ADEP output connector invokes the appropriate Document Services Process and sends the documents in the request.
4. The Document Services Process processes the documents and delivers the final output.

Creating an output connector enabled Document Services Process

You create and activate the process as described in the Document Services Workbench documentation. To enable the ADEP output connector to invoke the deployed service, you must add the input variables below to the Document Services Process.

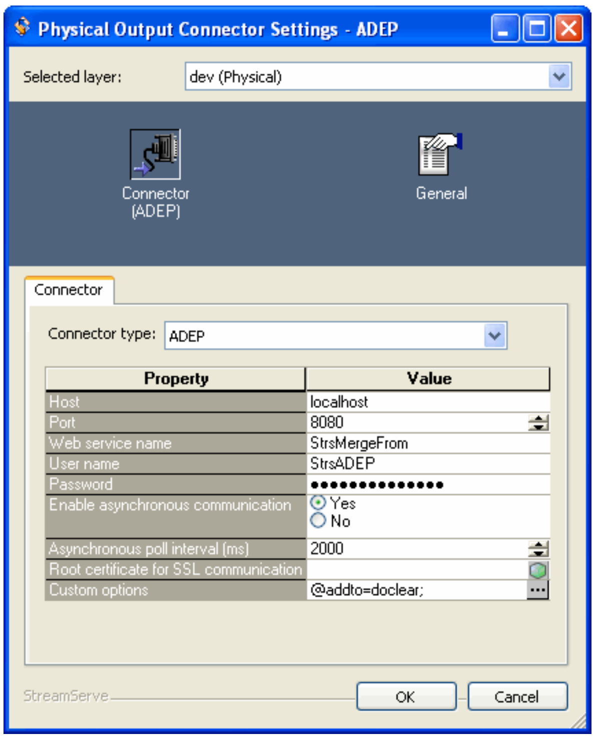
Variable name	Type	Comment
`inputDoc`	`document`	Mandatory
`optionsMap`	`map`	Optional. Used if custom keys are specified in the ADEP filter settings.

Only those Document Services Processes that follow this interface can be invoked by an ADEP output connector.

Creating an ADEP output connector

You create an ADEP output connector the same way as you create other output connectors in Design Center.

Connector settings



Selected layer: dev (Physical)

Connector type: ADEP

Property	Value
Host	localhost
Port	8080
Web service name	StrsMergeFrom
User name	StrsADEP
Password	●●●●●●●●●●●●●●●
Enable asynchronous communication	[✓ Yes] No
Asynchronous poll interval (ms)	2000
Root certificate for SSL communication	
Custom options	@addto=doclear;

Figure 3 Physical Output Connector Settings dialog box, ADEP connector

Setting	Description
Host	The host name or IP address of the server hosting ADEP. For example: `localhost`
Port	The port used by the ADEP server. For example: `8080`
Web service name	The name (case sensitive) of the service to invoke. This name must be the same as the corresponding process created in the Document Services Workbench.

Setting	Description
User name	User name to connect to the server hosting ADEP. Used in case of basic HTTP authentication.
Password	Password to connect to the server hosting ADEP. Used in case of basic HTTP authentication.
Enable asynchronous communication	**Yes** – Make asynchronous calls to the service. This option is used when invoking long-lived ADEP services. **No** – Make synchronous calls to the service. This option is used when invoking short-lived ADEP services.
Asynchronous poll interval	Only used together with asynchronous calls. This is the interval (milliseconds) used to check for a response to the invocation request.
Root certificate for SSL communication	The root certificate used when HTTPS is used as web service protocol (secure communication). The certificate must be available from a resource set connected to the Platform.
Custom options	A list of custom keys (key-value pairs) to include in the invocation request. To be able to handle custom keys, the service must have a variable named `optionsMap` of the type `map`. All custom keys defined here will be added to the `optionsMap` variable in the invoked service. The values provided can be extracted in the receiving Document Services Process by using an XPath expression in the Document Services Process. Examples of custom keys are passwords for creating password encrypted PDF files. For example: **Key:** `pdfpassword` **Value:** `encrypted`

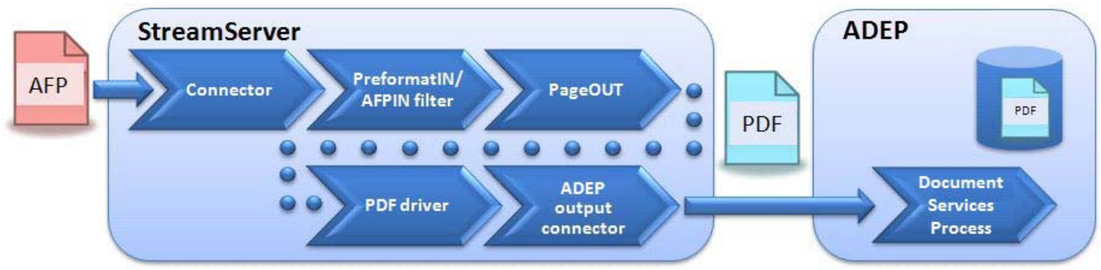
Usage scenario

Background

A business process requires that an AFP file of invoices be converted to PDF, and then passed to ADEP for storage.

Actions

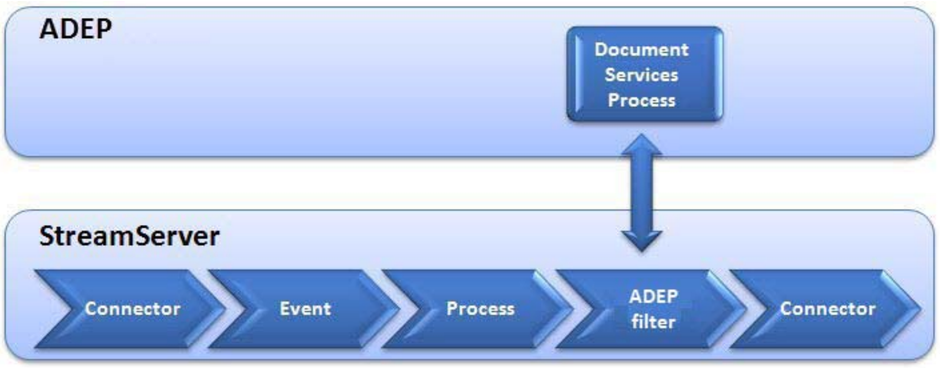
StreamServer is added to the pipeline. StreamServer retrieves the AFP input via an input connector, an AFPIN filter, and a PreformatIN Event. The AFP data is then transformed to PDF data via a PageOUT Process and a PDF driver. The PDF output is finally passed on to a Document Services Process via an ADEP output connector.



ADEP filter

The ADEP filter is used when StreamServer delivers the final output.

Example – ADEP filter usage



1 StreamServer receives input via an input connector.

2 StreamServer uses the appropriate Event/Process configuration to create documents.

3 The ADEP filter invokes the appropriate Document Services Process and sends the documents in the request.

4 The Document Services Process processes the documents, and sends the processed documents in the response to StreamServer.

5 StreamServer delivers the final output via an output connector.

If the web service goes down before the documents are sent in the web service response, no output is delivered. In this case an error message is logged.

Creating a filter enabled Document Service Process

You create and activate the process as described in the Document Services Workbench documentation. To enable the ADEP filter to invoke the deployed service, you must add the following input and output variables to the Document Services Process.

Input/output	Variable name	Type	Comment
Input	`inputDoc`	`document`	Mandatory
Input	`optionsMap`	`map`	Optional. Used if custom keys are specified in the ADEP filter settings.
Output	`outputDoc`	`document`	Mandatory

Only those Document Services Processes that follow this interface can be invoked by an ADEP filter.

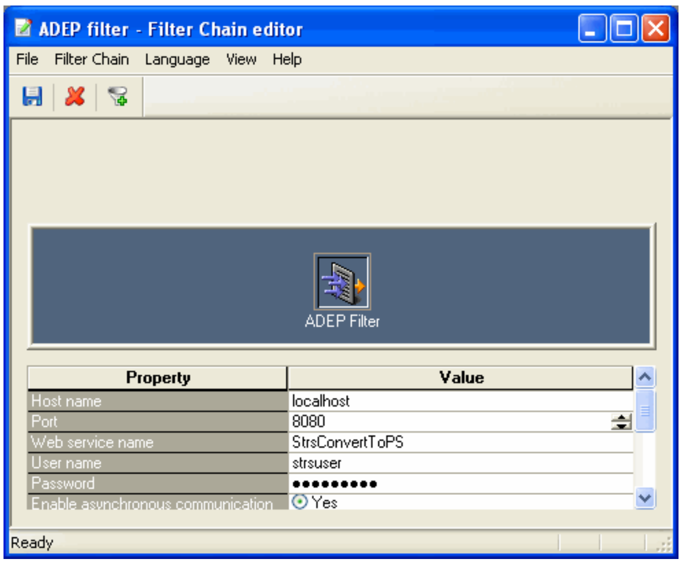
Creating an ADEP filter

You create an ADEP filter the same way as you create other output filters in Design Center. This means you must create a filter chain resource, create and configure the ADEP filter in the filter chain, and connect the filter chain to the appropriate output connector.

To create and apply an ADEP filter

1. Create a Filter Chain resource in a resource set connected to the Platform.
2. Add an ADEP filter to the filter chain.
3. Configure the filter (see *Filter settings* below) and save the Filter Chain resource.
4. Add the Filter Chain to the appropriate output connector.

Filter settings



Property	Value
Host name	localhost
Port	8080
Web service name	StrsConvertToPS
User name	strsuser
Password	●●●●●●●●●
Enable asynchronous communication	Yes

Figure 4 Filter Chain editor, ADEP filter

Setting	Description
Host name	The host name or IP address of the server hosting ADEP. For example: `localhost`
Port	The port used by the ADEP server. For example: `8080`

Setting	Description
Web service name	The name (case sensitive) of the service to invoke. This name must be the same as the corresponding process created in the Document Services Workbench.
User name	User name to connect to the server hosting ADEP. Used in case of basic HTTP authentication.
Password	Password to connect to the server hosting ADEP. Used in case of basic HTTP authentication.
Enable asynchronous communication	**Yes** – Make asynchronous calls to the service. This option is used when invoking long-lived ADEP services. **No** – Make synchronous calls to the service. This option is used when invoking short-lived ADEP services.
Asynchronous poll interval	Only used together with asynchronous calls. This is the interval (milliseconds) used to check for a response to the invocation request.
Root certificate for SSL communication	The root certificate used when HTTPS is used as web service protocol (secure communication). The certificate must be available from a resource set connected to the Platform.
Custom options	A list of custom keys (key-value pairs) to include in the invocation request. To be able to handle custom keys, the service must have a variable named `optionsMap` of the type `map`. All custom keys defined here will be added to the `optionsMap` variable in the invoked service. The values provided can be extracted in the receiving Document Services Process by using an XPath expression in the Document Services Process. Examples of custom keys are passwords for creating password encrypted PDF files. For example: **Key:** `pdfpassword` **Value:** `encrypted`

Usage scenario

Background

A StreamServer user needs to encrypt PDF documents, but the encryption cannot be done using StreamServer functionality. This functionality can be leveraged by invoking a Document Service Process to encrypt the PDF documents.

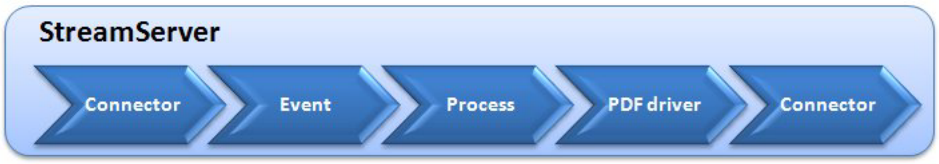


Figure 5 Before ADEP filter

Actions

An ADEP filter is added after the PDF driver. The ADEP filter sends the formatted PDF documents to a Document Services Process. The Document Services Process encrypts the documents, and returns the encrypted documents to StreamServer via the ADEP filter.

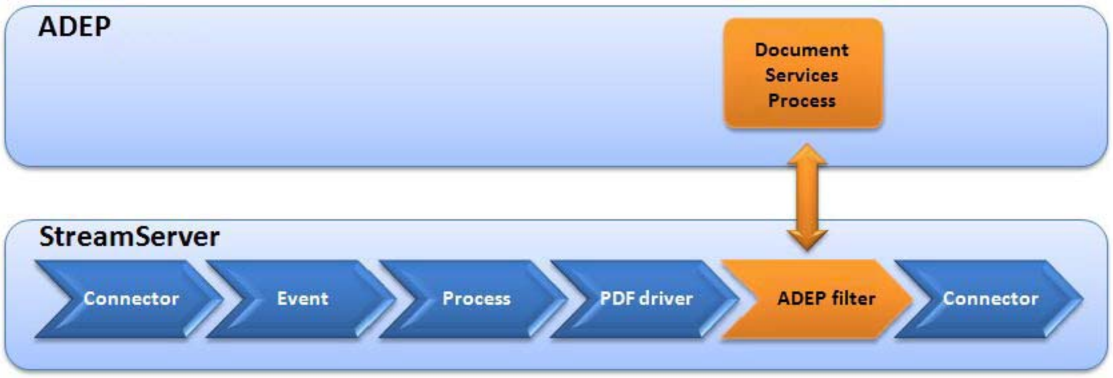


Figure 6 After ADEP filter

Global ADEP filter and connector settings

Apart from the settings configured in the ADEP filter and ADEP output connector GUI, you may need to change some of the global settings in the configuration file `strslcfilter.config.xml`. This configuration file is located in:

`<StreamServe_installation>\Services\XFA\<Version>\Service`

```
<?xml version="1.0" encoding="utf-8"?>
<lcfilter>
  <setting key="maxinlinesize">65536</setting>
  <setting key="timeout">120</setting>
  <setting key="retries">5</setting>
 </lcfilter>
```

Figure 7   strslcfilter.config.xml – example

These settings apply to all ADEP filters and ADEP output connectors.

Key	Description
`maxinlinesize`	The maximum size (bytes) allowed for a document to make it base64 encoded inline. If this size is exceeded, the document will be stored as a DIME attachment instead.
`timeout`	The time (seconds) to wait for a response to the request. If this time is exceeded, the connection is closed.
`retries`	The number of retry attempts in case of communication errors.

Sample Projects

The installation media includes two pre-configured sample Projects:

- Basic sample Project – The `SampleProject.dcpackage` Project shows how ADEP and Production Print processes can be integrated. The provided example integrations can be used as a pattern for your implementation.
- XFA chart sample Project – The `Xfacharts.dcpackage` Project shows how StreamServe StoryTeller can be used to insert dynamic business graphics (such as charts) into documents created using the XFA processor.

Why sample Projects

The sample Projects can be used:

- For educational purposes.
- To verify that Production Print has been properly installed and configured.

Where to find sample Projects

The sample Projects are provided on the installation media in the folder `Extras\sampleproject`.

By default, when installing **Design Center** from the Production Print installation media, the sample Projects are also installed in:

```
C:<StreamServe installation>\Services\XFA\<Version>\Tool
```

Prerequisites

- To run a sample Project, the sample Project's processes and resources must be installed.
- If you are running an ADEP server (i.e. invoking StreamServer applications from ADEP), the Production Print DSC must be installed.

For more information, see the *Adobe Digital Enterprise Platform, Extension – Production Print, Installation Guide*.

In this chapter

Running sample Projects

You must unpack a sample Project before you can export and deploy it.

To unpack a sample Project file

1. Open Design Center.
2. Select **File** > **Unpack Project**.
3. Browse to and open the package file. The Unpack Project dialog opens.
4. Specify where to unpack the Project files and click **OK**.

To export and deploy a sample Project

In Design Center, export the Project. See the *StreamServe Persuasion SP5 Design Center* documentation.

To deploy a sample project

1. Open Control Center.
2. Create the StreamServer application to deploy the sample Project to.
3. Deploy the sample Project to the StreamServer application.

See the *StreamServe Persuasion SP5 Control Center* documentation.

To start the StreamServer application

In Control Center, right-click the StreamServer application and select **Start**.

You also stop and redeploy StreamServer applications from Control Center. See *StreamServe Persuasion SP5 Control Center* documentation.

Verifying the basic installation

You can use the `SampleProject.dcpackage` Project to verify that Production Print has been properly installed and configured.

To verify the basic functionality of the installation

1 Unpack, export, and deploy the `SampleProject.dcpackage` Project. See *Running sample Projects* on page 58.

2 Create a folder `input` in the directory `C:\ManagementGateway\1.0\root\applications\<`*`your app name`*`>\Dev`

3 Copy the file `Purchase Order.xml` from the directory `C:\ManagementGateway\1.0\root\applications\<`*`your app name`*`>\data\samples` to the `input` folder.

4 Read the log in Control Center to verify that the file has been processed.

5 Verify that you have a file named `purchaseorder.pdf` in the directory `C:\ManagementGateway\1.0\root\applications\<`*`your app name`*`>\Dev\output`

To verify the AFP to PDF sample Project

1 Create a folder `input2` in the directory `C:\ManagementGateway\1.0\root\applications\<`*`your app name`*`>\Dev`

2 Copy the file `purchaseorder.afp` from the directory `C:\ManagementGateway\1.0\root\applications\<`*`your app name`*`>\data\samples` to the `input2` folder.

3 Read the log in Control Center to verify that the file has been processed.

4 Verify that you have five files named `<`*`Number`*`>.pdf` in the directory `C:\ManagementGateway\1.0\root\applications\<`*`your app name`*`>\Dev\output` where `<`*`Number`*`>` is the PO number from the input data.

Basic sample Project configurations

The `SampleProject.dcpackage` sample Project shows how ADEP and Production Print processes can be integrated. The provided example integrations can be used as a pattern for your implementation.

Sample 1 configuration – XML Input Data

The Sample 1 configuration illustrates a basic use case with synchronous post-processing, including document sorting and OMR marking.

You can connect the Process to the `PDF encrypt` output connector, install the sample LCA package, and configure the filter on the connector to point to the computer where the LCA is installed. This illustrates how Production Print can connect to an ADEP server, process a PDF, and return the PDF to Production Print.

Sample 2 configuration – ASCII Input Data

The Sample 2 configuration illustrates how to bind ASCII data to a form template.

1. Copy the file `Invocie.grb` from the samples directory to the input directory.
2. Verify that the `output.pdf` file is created in the output directory.

Sample 3 configuration – Dunning Notice Process

The Sample 1 and Dunning Notice Process configurations can be used together to illustrate remote invocation of Production Print from ADEP.

1. Install and deploy the LCA package on an ADEP server.
2. Invoke the `DunningNoticeStage` service with the `DunningNotice.xml` file as input (available in resources). The XML file and the `DunningNotice.xdp` are sent to Production Print.

   The data file and the template will be merged and stored in the Production Print Post-processor repository.
3. Invoke the `DunningNoticePrint` service with the file `lcpp.ppq` (post-processor query file) as input. The PPQ file is sent to Production Print that will get the staged Dunning Notice from the Post-processor repository, format it as a PDF file, and write it to the output folder as `output.pdf`.

Sample 4 configuration – Dunning Notice Generate

The Dunning Notice Generate configuration illustrates how Production Print can be invoked with data and a template.

Production Print will merge the data and the template and return the result as an AFP file to ADEP.

1. Invoke the `DunningNoticeGenerate` service with the `Dunning Notice.xml` file as input. The resulting AFP file will be written to a file on the ADEP server (default `c:\result.afp`).

Sample 5 configuration – AFP2PDF

The Sample 5 configuration illustrates how AFP files can be converted to PDF files using the PreformatIN tool and dynamic overlays in the PageOUT Process.

The PreformatIN Event will read the `PurchaseOrder.afp` file through an AFP2LXF filter and apply a pattern match to identify the document type. The PageOUT Process will apply the dynamically generated LXF pages to a logical page each. The result will be rendered as PDF using a driver on the output connector.

1 Invoke the AFP2PDF configuration by submitting the `purchaseorder.afp` file to the `Watchfolder2` input directory. The result is five PDF files in the output directory, one for each document in the AFP file.

Platform configuration

The Platform in the sample Project contains input and output connectors in the physical layer named `Dev`.

Input connectors

`input`	Directory scanning connector. The scanned folder is `.\input` (relative to the working directory of a deployed Project).

`DunningNoticeStage`
`DunningNoticePrint`
`DunningNoticeGenerate`
	Service Request connectors. Expose Message configurations as services through the StreamServe Services Gateway.

`WatchFolder2`	Directory scanning connector. The scanned directory is `.\input2` (relative to the working directory). The scanned file type is `*.afp`. The AFP2LXF filter is applied in the input pipeline of the connector.

To view the connector settings

1 Activate the Platform view.

2 Right-click the connector and select **Settings**.

3 Switch between the logical and physical layer in the Settings dialog box to view all connector settings.

Output connectors

`PDF`	File connector. The output file is set to `.\output\purchaseorder.pdf` (relative to the working directory of a deployed Project). The driver device is set to `PDF` with default options. Output mode is set to `Job` in order to keep all input in one output file.
`PDF encrypt`	File connector with the same settings as the `PDF` connector. The difference is that the `PDF encrypt` connector has a filter configuration that can invoke an ADEP service remotely, to encrypt and password protect the output data before it is written to file.
`PostProcessing`	File connector. The output file is set to `.\output\output.pdf` (relatively to the working directory of a deployed Project). The driver device is set to `PDF` with default options. Output mode is set to `Job` in order to keep all input in one output file.
`PPRepository`	Post-processor repository connector. Stores output in the embedded Post-processing repository using the alias `lcpp`. The driver device is set to `SDR`.
`LC Response`	A Null connector with the option **Include result in service response** set. The driver device is set to `AFP`.
`PDF Bypass`	Similar to the PDF connector, but the output mode is set to Process.

Resource set

The default resource set in the sample Project contains three folders, one for each Message type:

- `Invoice`
- `Purchase Order`
- `Dunning Notice`

There are sample resources for the configuration of the Messages:

- `/Invoice/invoice.grb` is a text print file that is input to the Message Sample 2 – ASCII Input Data.
- `/Invoice/invoice.xdp` is the template used by the Message Sample 2 – ASCII Input Data.

- `/Purchase Order/Purchase Order.xml` is input to the Message Sample 1 – XML Input Data.
- `/Purchase Order/Purchase Order.xdp` is a pre-defined form template for use with the Message Sample 1 – XML Input Data.
- `/Dunning Notice/Dunning notice.xml` is input to the Messages Sample 3 – Invocation by ADEP and Sample 4 – Dunning Notice Generate.
- `/AFP2PDF/AFP2PDF`, `f`ilter pipeline with the AFP2LXF filter configured.
- `/AFP2PDF/purchaseorder.afp` sample input file for the Sample 5 AFP2PDF configuration.

Message configurations

Sample 1 – XML Input Data (using an existing form template)

This Message uses a static template (`purchase order.xdp`). The form template uses an XML data file as input.

The Message has an Event and a Process configured. The Event uses a pattern to detect the data file type for the Message. The pattern is `/batch` to match the root node of the `Purchase Order.xml` data file. The Process links to a form template when the input data matches the pattern of the Event. The form template `Purchase Order.xdp` from the resource set is loaded in the Process.

Settings:

- Record mode is used with the record trigger `transaction`.
- StreamServe variable mapping is used to map the SOM Expression `$record.header.txtPONum` to the StreamServe variable `$ponum`. The variable is used in the Runtime settings to sort the forms in the batch input file.

Sample 2 – ASCII Input Data (using a template created from scratch in Design Center)

The input data to the form template is a text file. The Event uses the PageIN tool to detect the data file type and to extract the content of the data file into a StreamServe Message.

The sample file `invoice.grb` (from the resource set) is used to configure the Event. A pattern is used to detect the string `INVOICE` in a set of coordinates in the page. The Field Tool in PageIN has been used to extract data from coordinates on the page to the Message.

The Process uses the setting **Add the Message as a Data Connection in the Data View** (See *Settings dialog box* on page 86) to present the content of the Message in the ADEP Designer Data View. The content of the Data View has been used to create bindings to the fields in the form template.

Sample 3 – Invocation by ADEP (remote invocation of Production Print)

The Input data to the XMLIN Event is the XML file `Dunning Notice.xml`. The pattern is set to detect the element `<transaction>`. The process is configured to load a template dynamically. The variable `$template` will be assigned a template from the input connector automatically.

Sample 4 – Dunning Notice Generate (remote invocation of Production Print)

This Message configuration is identical to Sample 3. The difference is how it is used in the runtime.

Sample 5 - AFP2PDF

This Message configuration uses PreformatIN and PageOUT to convert AFP files to PDF files using the AFP2LXF filter in the platform.

Runtime configurations

Job Purchase Order

This job configuration uses the Sample 1 Message. The Event is connected to the `WatchFolder` input connector.

The output connector selection method is set to Static and it uses the `PostProcessing` output connector. The connector settings for the `PostProcessing` output connector uses:

- OMR marking on the Process Begin tab.
- Document sorting is based on the `$ponum` variable. Sorting is set to descending order.

Job Invoice

This job uses the Sample 2 Message. The Event is connected to the `WatchFolder` input connector.

The output connector selection method is set to **Static** and it uses the `PDF` output connector.

Job Dunning notice Stage

This job uses the Sample 3 Message. The Event is connected to the `DunningNoticeStage` input connector.

The output connector selection method is set to **Static** and it uses the `PPRepository` output connector.

Job Dunning notice Process

This is a Post-processor repository configuration. The job is connected to the `DunningNoticePrint` input connector.

The output connector selection method is set to **Static** and uses the `PDF` output connector.

Job Dunning notice Generate

This job uses the Sample 4 Message. The Event is connected to the `DunningNoticeGenerate` input connector.

The output connector selection method is set to **Static** and it uses the `LC Response` output connector.

Job AFP2LXF

This job sets the Runtime connector settings of the `PDF Bypass` connector to use a variable to produce unique file names for PDF files.

XFA chart sample Project configurations

The `Xfacharts.dcpackage` sample Project shows how the StreamServe StoryTeller Process tool can be used to include dynamic business graphics, such as charts, into documents created using the XFA processor.

Platform configuration

The Platform configuration in the XFA chart sample Project contains the input and output connectors below.

Input connectors

`in`	Directory scanning connector. The main input connector, receiving input in XML format. The scanned folder is `.\input` (relative to the working directory of the deployed Project).
`ppq`	Directory scanning connector. A secondary input connector that processes the PPQs (post-processor queries) created by the first job. The scanned folder is `.\input\ppq` (relative to the working directory of the deployed Project).

Output connectors

`PDF`	File connector. The main output connector, writing the final output in PDF format. The output file is set to `.\output\out.pdf` (relative to the working directory of the deployed Project). The driver device is set to `PDF` with default options.
`LXF`	File connector. Generates the charts from the StoryTeller Process. The output file is set to `..\data\overlays\chart.lxf`. The driver device is set to `LXF` with default options.
`PPR`	Post-processor repository connector. This connector stages the XFA output while the charts are being generated. Output is stored in the embedded Post-processing repository using the alias `xfacharts`. The driver device is set to `SDR`.

Message configuration

This Message uses a static template (`Purchase Order.xdp`) from the resource set. The form template uses an XML data file as input.

The Message configuration contains one Event and two Processes:

- `PO` – The Purchase Order XMLIN Event that parses the XML input and creates a Message.
- `Chart` – The StoryTeller Process that creates a dynamic chart object based on the XML input.
- `PO` – The Purchase Order ADEP Designer Process that merges the XML input with the XFA template.

Runtime configuration

The Runtime includes two job configurations.

- `Job` – This job connects the `in` input connector with the `PO` Event. The `Char`t Process is connected to the `LXF` output connector, and the `PO` Process is connected to the `PPR` output connector. For both connectors, the output connector selection method is set to **Static**.
- `New Post-processor 1` – This job connects to the `ppq` input connector and the output is written to the `PDF` output connector. The output connector selection method is set to **Static**.

How it works

1. The `Purchase Order.xml` resource is copied to the `input` folder.
2. The `in` input connector detects the file and creates an input job.
3. The file is split into four Events. The data for each Event is extracted into a Message by the `PO` Event.
4. The `Chart` Process is executed for every Message. The Process creates a chart by merging the Message data with the StoryTeller template. The result is written to the `chart.lxf`. overlay file by the `LXF` output connector.
5. The `PO` Process is executed for every Message. The Process creates a transaction document by merging the XML input with the XFA template. The result is written to a Post-processor repository by the `PPR` connector. When the last file is written to the repository, an `xfachart.ppq` file is created by the `PPR` output connector.
6. The `ppq` input connector reads the PPQ file and performs the query against the Post-processing repository. The resulting documents are retrieved and sent to the Post-processor.
7. For the first page in each document retrieved, the Post-processor inserts the corresponding LXF overlay created in the first job. The result is sent through the PDF driver and is delivered by the `PDF` output connector.

Label printer support

Production Print has support for the following label printers:

- ZPL II – see *ZPL II* on page 70 for details.
- Intermec FP/DP – see *Intermec FP/DP* on page 72 for details.
- Intermec IPL – see *Intermec IPL* on page 75 for details.
- Printronix PGL/IGP – see *Printronix PGL/IGP* on page 78 for details.
- TEC – see *TEC* on page 80 for details.

Notes about label printer support

Label Printer languages has limitations in object and object property support when compared to formats such as PDF, PCL, AFP, PS, etc.

For performance reasons, Production Print does not provide the same WYSIWYG support as preview in ADEP Designer. For example:

- Gradient fills are not rasterized.
- Line styles, such as dashed, are not supported for all printer languages.

ZPL II

Text output – ZPL II

Scalable fonts

The following scalable fonts are supported:

- CG Triumvirate Bold Condensed

This is the only font that provides WYSIWYG to the user.

Adobe fonts

All Adobe fonts are mapped to the above font.

ASCII

Support for US ASCII table only.

Barcode support – ZPL II

The following barcodes are supported:

- Aztec
- Codabar
- Code 11
- Code 128
- Code 2 of 5 Industrial
- Code 2 of 5 Interleaved
- Code 2 of 5 Standard
- Code 3 of 9
- Code 49
- Code 93
- Data Matrix
- EAN13
- EAN8
- Logmars
- MSI
- PDF417
- Planet Code
- Plessey
- QR Code
- RSS14
- RSS14 Expanded
- RSS14 Limited
- RSS14 Stacked
- RSS14 Stacked Omnidirectional
- RSS14 Truncated
- UPC-A
- UPC-E
- UPS Maxicode
- US Postal DPBC
- US Postal Standard
- US Postal Zip-5

RFID support – ZPL II

RFID barcode is supported.

Air protocol	EPC Class 1 Generation 2.
Barcode value	Treated as HEX96.

GUI objects support – ZPL II

Support for GUI objects is listed below.

Object	**Comment**
Vertical lines	Supported.
Horizontal lines	Supported.
Diagonal lines	Supported.
Rectangles	Supported.
Circles	Supported.
Images	Supported.
Line style	Only solid line supported.
Rotation	Supports 90, 180, and 270 degrees rotation.

Intermec FP/DP

Text output – Intermec FP/DP

Scalable fonts

The following scalable fonts are supported:

- Century Schoolbook BT
- Dutch 801 Roman BT
- Letter Gothic 12 Pitch BT
- Monospace 821 BT
- OCR-B 10 Pitch BT
- Swiss 721 Bold BT
- Swiss 721 Condensed BT
- Dutch 801 Bold BT
- Futura Light BT
- Monospace 821 Bold BT
- OCR-A BT
- Prestige 12 Pitch Bold BT
- Swiss 721 BT
- Zurich Extra Condensed Bold

These are the only fonts that provide WYSIWYG to the user.

Adobe fonts

All Adobe fonts are mapped to the above fonts (fonts with similar metrics).

ASCII

Support for US ASCII table only.

Barcode support – Intermec FP/DP

The following barcodes are supported:

- Aztec
- Codabar
- Code 11
- Code 128
- Code 2 of 5 Industrial
- Code 2 of 5 Interleaved
- Code 2 of 5 Matrix
- Code 2 of 5 Standard
- Code 3 of 9, Code 93
- Code 49, MSI, Planet Code
- Data Matrix
- EAN13
- EAN8
- PDF417
- Plessey
- QR Code
- RSS14
- RSS14 Expanded
- RSS14 Limited
- RSS14 Stacked
- RSS14 Stacked Omnidirectional
- RSS14 Truncated
- UPC-A
- UPC-E, UPS Maxicode
- US Postal DPBC
- US Postal Standard
- US Postal Zip-5

RFID support – Intermec FP/DP

RFID barcode is supported.

Air protocol	EPC Class 1 Generation 2.
Barcode value	Treated as HEX96.

GUI objects support – Intermec FP/DP

Support for GUI objects is listed below.

Object	Comment
Vertical lines	Supported.
Horizontal lines	Supported.
Diagonal lines	Not supported.
Rectangles	Supported.
Circles	Not supported.
Images	Supported.
Line style	Only solid line supported.
Rotation	Supports 90, 180, and 270 degrees rotation.

Intermec IPL

Text output – Intermec IPL

Scalable fonts

The following scalable fonts are supported:

- Century Schoolbook BT
- Dutch 801 Roman BT
- Letter Gothic 12 Pitch BT
- Monospace 821 BT
- OCR-B 10 Pitch BT
- Swiss 721 Bold BT
- Swiss 721 Condensed BT
- Dutch 801 Bold BT
- Futura Light BT
- Monospace 821 Bold BT
- OCR-A BT
- Prestige 12 Pitch Bold BT
- Swiss 721 BT
- Zurich Extra Condensed Bold

These are the only fonts that provide WYSIWYG to the user.

Adobe fonts

All Adobe fonts are mapped to the above fonts (fonts with similar metrics).

ASCII

Support for US ASCII table only.

Barcode support – Intermec IPL

The following barcodes are supported:

- Aztec
- Codabar
- Code 11
- Code 128
- Code 2 of 5 Industrial
- Code 2 of 5 Interleaved
- Code 3 of 9
- Code 49
- Code 93
- Data Matrix
- EAN13
- EAN8
- PDF417
- Planet Code
- QR Code
- RSS14
- RSS14 Expanded
- RSS14 Limited
- RSS14 Stacked
- RSS14 Stacked Omnidirectional
- RSS14 Truncated
- UPC-A
- UPC-E
- UPS Maxicode
- US Postal DPBC
- US Postal Standard
- US Postal Zip-5

RFID support – Intermec IPL

RFID barcode is supported.

Air protocol	EPC Class 1 Generation 2.
Barcode value	Treated as HEX96.

GUI objects support – Intermec IPL

Support for GUI objects is listed below.

Object	**Comment**
Vertical lines	Supported.
Horizontal lines	Supported.
Diagonal lines	Not supported.
Rectangles	Supported.
Circles	Not supported.
Images	Supported.
Line style	Only solid line supported.
Rotation	Supports 90, 180, and 270 degrees rotation.

Printronix PGL/IGP

Text output – Printronix PGL/IGP

Scalable fonts

The following scalable fonts are supported:

- CG Triumvirate Bold Condensed
- Courier Bold
- Letter Gothic Bold

These are the only fonts that provide WYSIWYG to the user.

Adobe fonts

All Adobe fonts are mapped to the above fonts (fonts with similar metrics).

ASCII

Support for US ASCII table only.

Barcode support – Printronix PGL/IGP

The following barcodes are supported:

- AUSPOST Custom 2
- AUSPOST Custom 3
- AUSPOST Replay Paid
- AUSPOST Standard
- Codabar
- Code 128
- Code 2 of 5 Industrial
- Code 2 of 5 Matrix
- Code 3 of 9
- Code 93
- Data Matrix
- EAN13
- EAN8
- MSI
- PDF417
- Planet Code
- Plessey
- UK/Royal Mail RM4SCCC
- UPC-A
- UPC-E
- UPS Maxicode
- US Postal DPBC
- US Postal Standard
- US Postal Zip-5

RFID support – Printronix PGL/IGP

RFID barcode is supported.

Air protocol	EPC Class 1 Generation 2.
Barcode value	Treated as HEX96.

GUI objects support – Printronix PGL/IGP

Support for GUI objects is listed below.

Object	**Comment**
Vertical lines	Supported.
Horizontal lines	Supported.
Diagonal lines	Supported.
Rectangles	Supported.
Circles	Supported.
Images	Supported.
Line style	Only solid line supported.
Rotation	Supports 90, 180, and 270 degrees rotation.

TEC

Text output – TEC

Bitmap fonts

The following bitmap fonts are supported:

- Courier
- Helvetica
- Letter Gothic
- OCR-A
- OCR-B
- Presentation
- Prestige Elite
- Times New Roman

These are the only fonts that provide WYSIWYG to the user.

Adobe fonts

All Adobe fonts are mapped to the above fonts (fonts with similar metrics).

ASCII

Support for US ASCII table only.

Barcode support – TEC

The following barcodes are supported:

- Code 128
- Code 2 of 5 Industrial
- Code 2 of 5 Interleaved
- Code 2 of 5 Matrix
- Code 3 of 9
- Code 93
- Data Matrix
- EAN13
- EAN8
- MSI
- PDF417
- QR Code
- RSS14
- RSS14 Expanded
- RSS14 Limited
- RSS14 Stacked
- RSS14 Stacked Omnidirectional
- UK/Royal Mail RM4SCCC
- UPC-A
- UPC-E
- UPS Maxicode
- US Postal DPBC
- US Postal Standard
- US Postal Zip-5

RFID support – TEC

RFID is not implemented.

GUI objects support – TEC

Support for GUI objects is listed below.

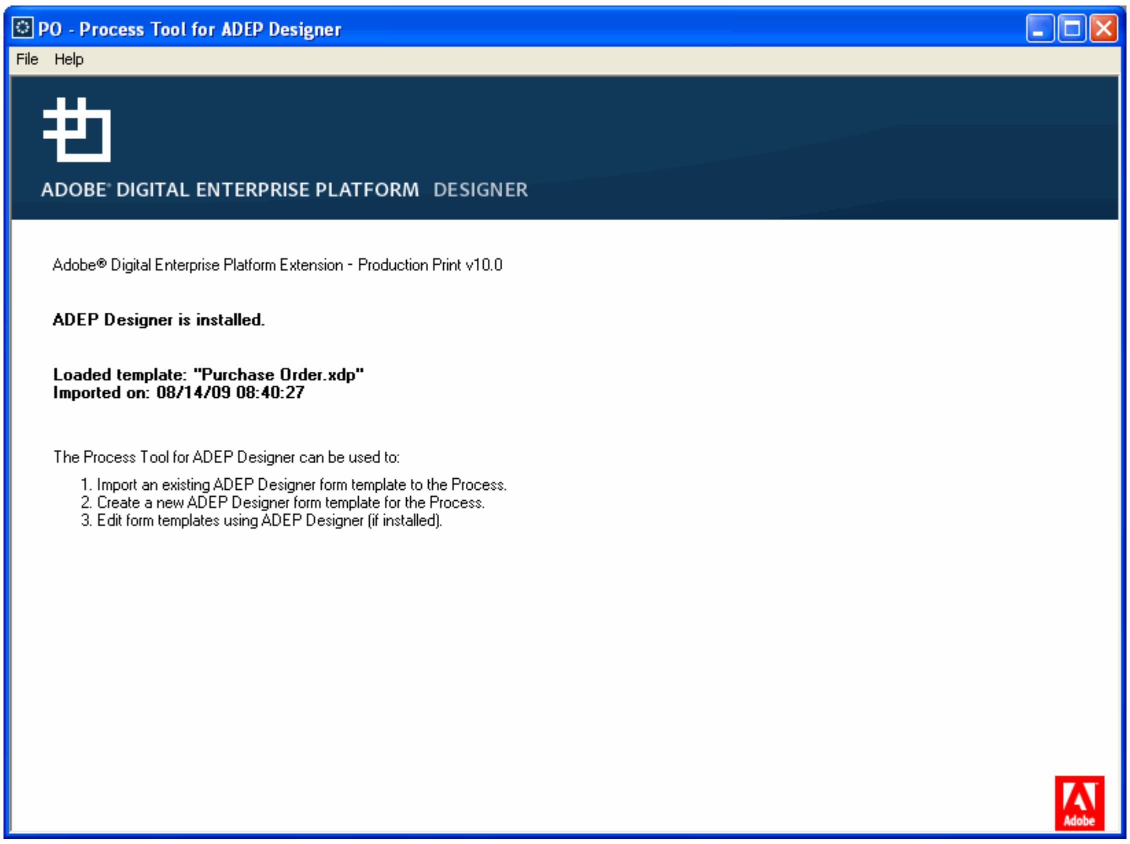
Object	Comment
Vertical lines	Supported.
Horizontal lines	Supported.
Diagonal lines	Supported.
Rectangles	Supported.
Circles	Supported.
Images	Supported.
Line style	Only solid line supported.
Rotation	Supports 90, 180, and 270 degrees rotation.

GUI reference

In this chapter

StreamServe Process Tool for ADEP Designer

The Process Tool start window is displayed when opening an ADEP Designer Process.


PO - Process Tool for ADEP Designer
File Help
ADOBE DIGITAL ENTERPRISE PLATFORM DESIGNER
Adobe

Adobe® Digital Enterprise Platform Extension - Production Print v10.0

ADEP Designer is installed.

Loaded template: "Purchase Order.xdp"
Imported on: 08/14/09 08:40:27

The Process Tool for ADEP Designer can be used to:

1. Import an existing ADEP Designer form template to the Process.
2. Create a new ADEP Designer form template for the Process.
3. Edit form templates using ADEP Designer (if installed).

Figure 8 The Process Tool for ADEP Designer start window

If a template is loaded, the name of the template and the date the template was imported are displayed.

If ADEP Designer is:

- Installed and you have chosen to load the ADEP Designer GUI at startup, the ADEP Designer GUI is launched.
- Not installed, you can use the **File** menu commands in this window to make your settings and select template.

In this chapter

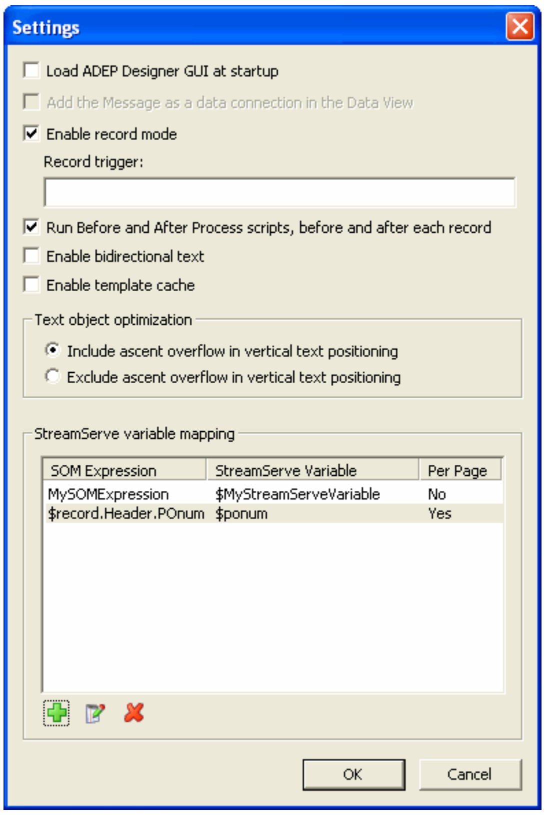
- *File menu commands* on page 85.
- *Settings dialog box* on page 86.
- *Select Template dialog box* on page 89.

File menu commands

Open/Select Template	Open the *Select Template dialog box*, where you choose template.
Save	Save the changes made.
Import LiveCycle Archive	Open the Select LCA archive to import dialog box, where you select LiveCycle Archive file (LCA) to import. The LCA file contains a complete package of a main XDP template and its dependencies. To be available for import into the Process tool, the LCA file has previously been imported into the Design Center as resource type **Sample**.
Export Message Schema	Create an XML schema file containing the StreamServe Message.
Export Preview XML	Create an XML file that contains the structure of the StreamServe Message, with sample data for each field.
Extract Resources to File	Open the Select Resource(s) to extract dialog box, where you can browse the resource set and select one or more resources to be extracted to file. The extracted resources are stored in the Process tool's current working directory, in the same structure as in the resource set, available to ADEP Designer.
Settings	Open the *Settings dialog box*.
Exit	Exit ADEP Designer. If there are modified and unsaved components, a dialog opens where you can specify what to save.

Settings dialog box

In the Settings dialog box you can define default settings and map SOM expressions with StreamServe variables.


Settings
Load ADEP Designer GUI at startup
Add the Message as a data connection in the Data View
Enable record mode
Record trigger:
Run Before and After Process scripts, before and after each record
Enable bidirectional text
Enable template cache
Text object optimization
Include ascent overflow in vertical text positioning
Exclude ascent overflow in vertical text positioning
StreamServe variable mapping
OK
Cancel

SOM Expression	StreamServe Variable	Per Page
MySOMExpression	$MyStreamServeVariable	No
$record.Header.POnum	$ponum	Yes

Figure 9 Settings dialog box

Load ADEP Designer GUI at startup

Select this option if you want to run the ADEP Designer GUI. The GUI opens automatically when you start the Process tool.

Note: If you have not installed ADEP Designer, this option is not available.

Add the Message as a data connection in the Data View

Select this option if you are creating a template from scratch. The StreamServe Message is added to ADEP Designer as a data connection.

Enable record mode

The Process can operate in two modes, record mode and non-record mode. In record mode, the data document is treated as a sequence of records. In the simplest case, each record in turn is loaded, processed, and unloaded before the next record is loaded. Record mode is provided purely as a way to reduce resource consumption (memory and CPU cycles) when dealing with large data documents.

Anything that can be done in record mode can also be done in non-record mode, provided that sufficient resources are available.

Record trigger

This field is enabled if **Enable record mode** is selected. It specifies how to divide the input data into several records by defining XML tag or level (a positive number).

Example 5 *Record trigger with the following input data structure*

```
<A>
 <B>
   <Data1>1</Data1>
   <Data2>2</Data2>
 </B>
 <C>
   <Data1>3</Data1>
   <Data2>4</Data2>
 </C>
 <B>
   <Data1>5</Data1>
   <Data2>6</Data2>
 </B>
</A>
```

If you set **Record trigger** to:

- `1`, there are three records.
- XML tag `B`, there are two records (the ones tagged `B`). The one tagged `C` is ignored.

  When using an XML tag as record trigger, deep-first search is executed. This means that the first XML tag found is used together with its level for the continued search. Only records tagged both with the defined XML tag and on the same level as the first found record are considered.

Run Before and After Process scripts, before and after each record

This field is enabled if **Enable record mode** is selected.

If this setting is selected, Before and After Process scripts will be executed before and after each record. If not selected, the before and after Process scripts will be executed before and after each Process.

For Projects created in Production Print, this functionality will be enabled by default.

For Projects created in previous versions than LiveCycle Production Print ES2, the behavior with scripts per Process will be default.

See *Using Before and After Process scripts* on page 106.

Enable bidirectional text

Select this option to enable bidirectional (BiDi) text. Bidirectional text can be used for Arabic and Hebrew notation. The **Enable Bidirectional Text** option is by default off.

Enable template cache

Select this option to enable caching of templates. This can significantly improve performance if you are using dynamic templates.

See *Template caching* on page 12.

Text object optimization

The font ascent value is the part of a character that extends above the baseline. This value is usually the same for all characters within a font. But if you use special characters, such as the Swedish characters `Ä` and `Å`, they might exceed the ascent value for the given font: ascent overflow. To avoid overlapping text lines, the ascent overflow value can be added to the text line height

- Include ascent overflow in vertical text positioning

  Ascent overflow is calculated, and the value is added to the text line height for the font. Since each character in the font is considered, this affects performance. Do not select this option if you do not use special characters. The **Include ascent overflow in text positioning** option is by default selected.

- Exclude ascent overflow in vertical text positioning

  Ascent overflow is not considered. If you use special characters text lines might overlap. Select this option if you prioritize performance.

StreamServe variable mapping

The list shows the mapping from a **SOM Expression** to a **StreamServe Variable**.

Per Page shows if the mapping will be done per document or per page:

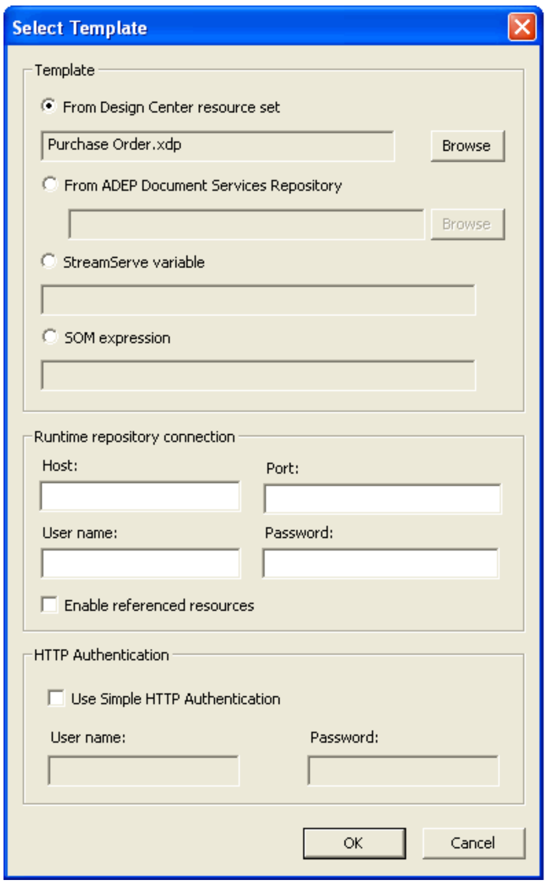
`Yes`	The mapping is done once before each page is output.
`No`	The mapping is done once before each document is output.

Mapping per page is specified when editing the mapping of variables in the Map Variables dialog box, option **Evaluate variable for each page**.

See *Get the value of a SOM expression to a StreamServe variable* on page 102.

Select Template dialog box

In the Select Template dialog box you select a template and specify its connection details.


Select Template

Template

- [✓ From Design Center resource set]: Purchase Order.xdp — Browse
- From ADEP Document Services Repository — Browse
- StreamServe variable
- SOM expression

Runtime repository connection

Host: Port:

User name: Password:

☐ Enable referenced resources

HTTP Authentication

☐ Use Simple HTTP Authentication

User name: Password:

OK Cancel

Figure 10 Select Template dialog box.

The settings are described in the table below:

Settings	**Description**
From Design Center resource set	Browse to and select a template contained in a resource set. The selected template will be statically associated with the Process.
From ADEP Document Services Repository	Browse to and select the template from the specified Document Services Repository. The repository connection and logon credentials specified in **Runtime repository connection** below will be used. The selected template will be loaded from the repository during runtime, when the process is started.

StreamServe variable	Specify a StreamServe variable pointing to a template. See *Using StreamServe variable to load template* on page 26.
SOM expression	Specify a SOM Expression in the data DOM pointing to a template. For example, `$record.templateLocation` See *Using SOM expression to load template* on page 28.
Runtime repository connection	Specify the connection profile (**Host**, **Port**, **User name**, and **Password**) for the Document Services Repository. **Enable referenced resources** – Enables references to resources in the Document Services Repository to be resolved. For example, references to fragments and images. The resources will be available in the ADEP Designer Process tool at design time and to the StreamServer application at runtime. If this option is cleared, only embedded resources are available. **Note:** This option makes the StreamServer application retrieve the template from the repository for each invocation, which may have negative impact on performance. The **Enable template cache** setting in the Settings dialog box has no impact if referenced resources is enabled.
HTTP Authentication	If using HTTP URI, you can select **Use Simple HTTP Authentication**. The specified **User name** and **password** will be used as authentication credentials.

Design Center

This chapter describes the Design Center commands specific for Production Print.

In this chapter

- *Tools menu commands* on page 91.
- *Resources menu commands* on page 91.
- *Dialog boxes* on page 92.

Tools menu commands

The Tools menu includes the commands described in the table below.

Select ADEP Document Services Repository connection	Open the *Select Active ADEP Document Services Repository Connection dialog box*, where you can manage connections to the Document Services Repository.

Resources menu commands

The Resources menu is available when the resource set view is active. It includes the commands described in the table below.

Import from ADEP Document Services Repository	Open the Select resource browser. Used to browse for and select a resource in the Document Services Repository. When a resource is selected, the *Import ADEP Document Services Repository Resource dialog box* opens.
Update all ADEP Document Services Repository Resources	Open the *Update all ADEP Document Services Repository resources dialog box*.

Dialog boxes

In this section

Select Active ADEP Document Services Repository Connection dialog box

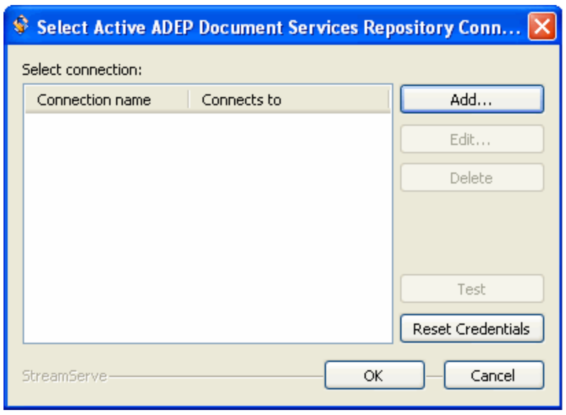


Figure 11 The Select Active ADEP Document Services Repository Connection dialog box.

This dialog provides a list of available connections to Document Services Repositories.

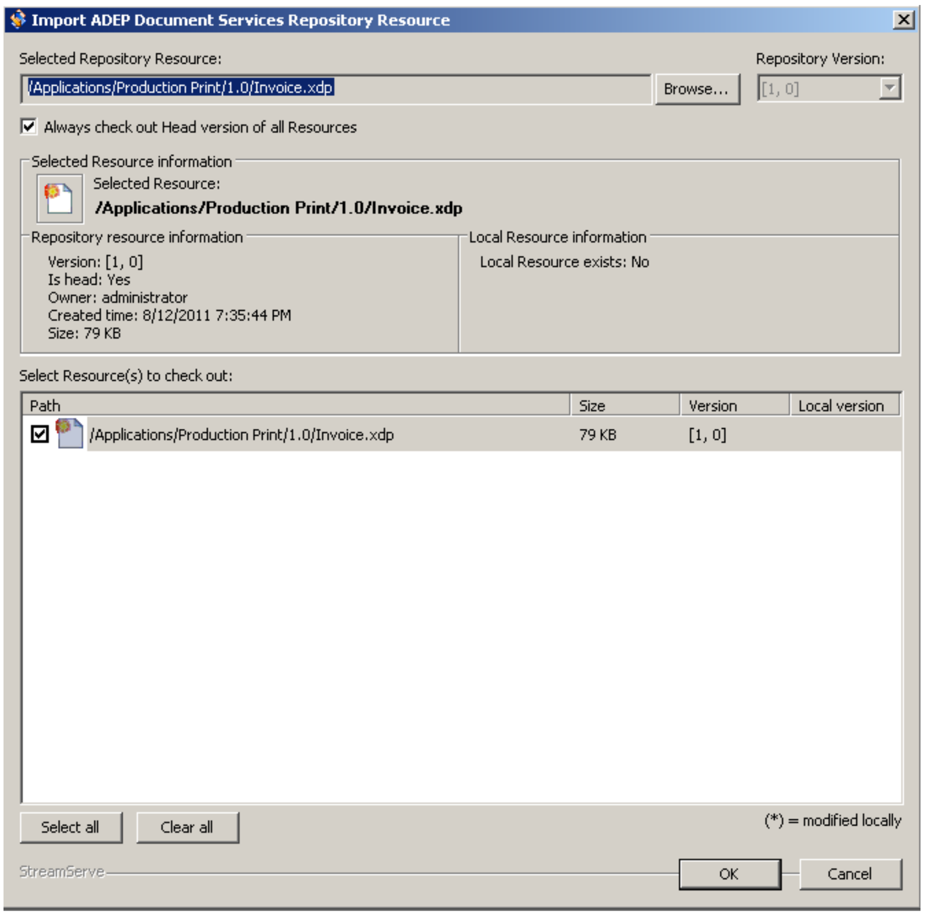
Note: You can define connections to several repositories, but you can only create and update resources from one repository at the time.

The connection selected is the one active.

Setting	Description
Select connection:	All defined connections to Document Services Repositories. • **Connection name** – the name of the connection. • **Connects to** – the name or IP address of the connected host and port number.

Setting	Description
Add	Add a new connection.
Edit	Edit an existing connection.
Delete	Delete selected (highlighted) connection.
Test	Check if the selected connection can be successfully established.
Reset Credentials	Clears your logon credentials to the Document Services Repository. Use this setting if user name of password have been changed on the server, or if you misspelled your password.

Import ADEP Document Services Repository Resource dialog box


Import ADEP Document Services Repository Resource
Selected Repository Resource:
Browse...
Repository Version:
Select all
Clear all
StreamServe
OK
Cancel

/Applications/Production Print/1.0/Invoice.xdp

[1, 0]

☑ Always check out Head version of all Resources

Selected Resource information
Selected Resource:
/Applications/Production Print/1.0/Invoice.xdp

Repository resource information
Version: [1, 0]
Is head: Yes
Owner: administrator
Created time: 8/12/2011 7:35:44 PM
Size: 79 KB

Local Resource information
Local Resource exists: No

Select Resource(s) to check out:

Path	Size	Version	Local version
☑ /Applications/Production Print/1.0/Invoice.xdp	79 KB	[1, 0]	

(*) = modified locally

Figure 12 The Import ADEP Document Services Repository Resource dialog box.

See *Icons used when accessing Document Services Repository* on page 99 for information on the document icons used in this dialog box.

The settings are described in the table below.

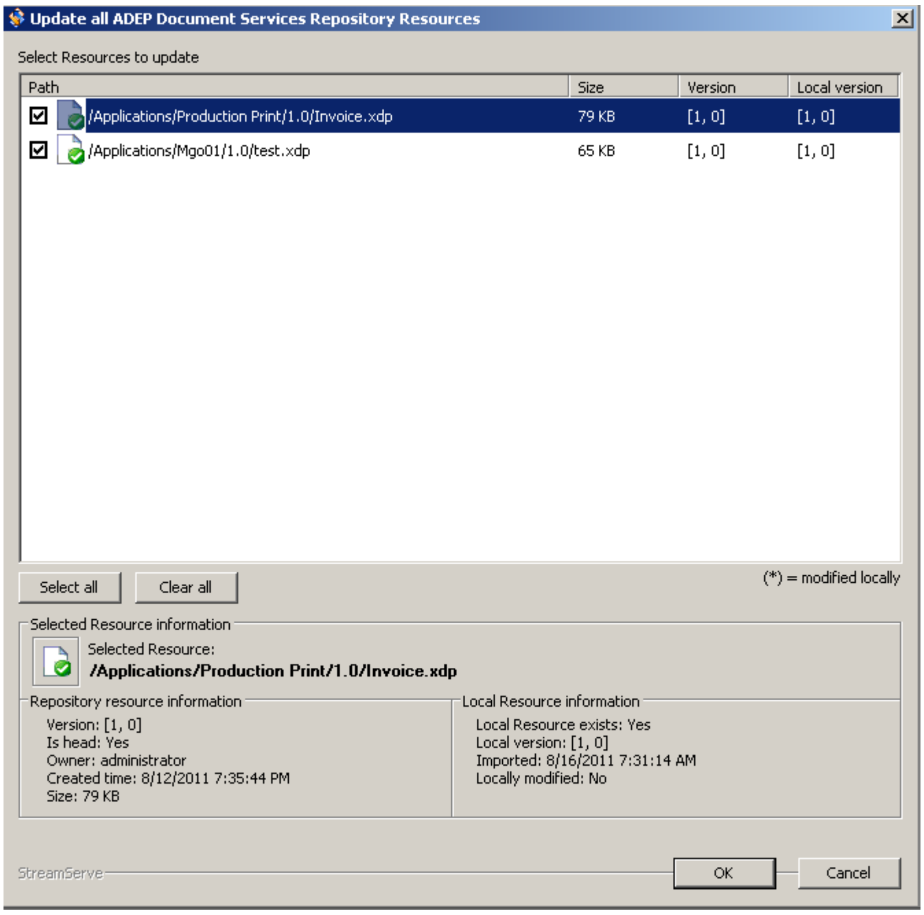
Setting	**Description**
Selected Repository Resource	Display path and file name of selected resource.
Repository Version	Resource version in the repository. The available versions can be selected from the drop-down list.
Always check out Head version of all resources	Import the latest version of the resource and its dependencies. This option is pre-selected.
Repository resource information	Display information about the selected (highlighted in the list) resource in the repository: **Version** – Version number of the selected resource. **Is head** – Yes/No Yes if current version is the most recent one. No if otherwise. **Owner** – Creator user name. **Created time** – Date and time the selected resource version was created. **Size** – Resource file size (KB). If the resource has been removed from the repository, information will be displayed about this.
Local Resource information	Display information about the selected (highlighted) resource in the local resource set: **Local resource exists** – Yes/No. Yes if the resource does already exist locally in the Design Center resource set. **Local version** – The version number of the resource stored on the locally in the Design Center resource set. **Imported** – Date and time the resource was imported and stored in the Design Center resource set. **Locally modified** – Yes/No. Yes if the resource has been changed locally.

Setting	Description
Select Resource(s) to check out	A list of selected repository resource and its dependencies. **Path** – Path and file name of the resource. **Size** – Resource file size (KB). **Version** – Version number of the resource in the repository. **Local version** – Version number of the resource stored in the Design Center resource set.
Select all	Select all objects in the list.
Clear all	Clear all check-boxes (i.e. select none of the objects in the list).

Update all ADEP Document Services Repository resources dialog box

This dialog box displays if a resource has been changed and needs to be updated.

When the dialog box opens, the resources that have an older version in the Design Center resource set than in the repository are pre-selected.


Update all ADEP Document Services Repository Resources

Select Resources to update

	Path	Size	Version	Local version
☑	/Applications/Production Print/1.0/Invoice.xdp	79 KB	[1, 0]	[1, 0]
☑	/Applications/Mgo01/1.0/test.xdp	65 KB	[1, 0]	[1, 0]

Select all | Clear all | (*) = modified locally

Selected Resource information

Selected Resource:
/Applications/Production Print/1.0/Invoice.xdp

Repository resource information
Version: [1, 0]
Is head: Yes
Owner: administrator
Created time: 8/12/2011 7:35:44 PM
Size: 79 KB

Local Resource information
Local Resource exists: Yes
Local version: [1, 0]
Imported: 8/16/2011 7:31:14 AM
Locally modified: No

StreamServe | OK | Cancel

Figure 13 The Update all ADEP Document Services Repository Resources dialog box

See *Icons used when accessing Document Services Repository* on page 99 for information on the document icons used in this dialog box.

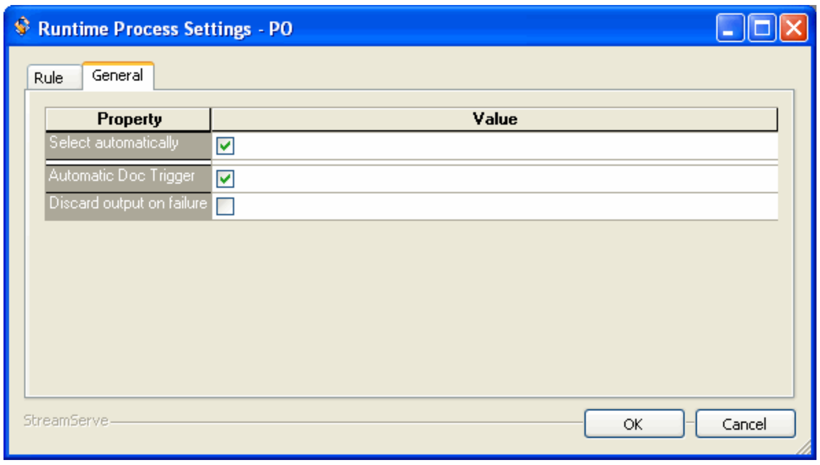
Setting	**Description**
Select Resources to update	**Path** – Path and file name of the resource. **Size** – Resource file size. **Version** – Version of the resource in the repository. **Local version** – Version of the resource stored in the Design Center resource set.
Select all	Select all objects in the list.
Clear all	Clear all check-boxes (i.e. select none of the objects in the list).
Selected Resource information	Information about the selected (highlighted) resource.

Setting	**Description**
Repository resource information	Display information about resource in the repository: **Version** – Version of the selected resource. **Is head** – **Yes**/**No** **Yes** if current version is the most recent one. **Owner** – Creator user name. **Created time** – Date and time the selected resource version was created. **Size** – Resource file size. If the resource has been removed from the repository, information will be displayed about this.
Local Resource information	**Local Resource exists** – **Yes**/**No**. **Yes** if the resource has been imported to the Design Center resource set. **Local version** – The version of the resource stored in the Design Center resource set. **Imported** – Date and time the resource was imported and stored in the Design Center resource set. **Locally modified** – **Yes**/**No**. **Yes** if the resource has been changed locally.

Runtime Process Settings dialog box – General tab

In the Runtime Process Settings dialog box, you can configure runtime specific settings for the selected Process.

On the **General** tab you specify general Runtime Process settings.



Property	Value
Select automatically	✓
Automatic Doc Trigger	✓
Discard output on failure	☐

Figure 14   Runtime Process Settings

Setting	Description
Select automatically	See standard *StreamServe Persuasion SP5 Design Center* documentation.
Automatic Doc Trigger	When selected, the **Document trigger variable** (specified in the Runtime Connector Settings dialog, Document Trigger tab) is disabled. This means that each record will be automatically mapped to one document. There will be no grouping of output into logical documents. This setting is by default selected for Projects upgraded from releases previous to LiveCycle Designer ES2. For new ADEP Designer Processes it is cleared by default.
Discard output on failure	Select to discard the output when an error occurs in the Process (e.g. a substitution cannot be found, or an XFA template contains errors). If selected, all output created by the Process (e.g. a PDF file) is discarded, and the job is marked as failed. If not selected, some output (or no output) is forwarded to the driver. The driver will try to process the output and, for example, create an empty PDF file or a PDF file that only contains the pages up to the page that failed. In both cases the job is marked as failed, and StreamServer will retry to process the job depending on the settings on the input queue.

The **Automatic Doc Trigger** setting is also available for StoryTeller Processes. Then, each process will be automatically mapped to one document.

Icons used when accessing Document Services Repository

The dialog boxes use a number of document icons:

The resource stored in the Document Services Repository has a newer version than the local resource (in the Design Center resource set). An import will overwrite the local resource with a newer version.

The resource does not exist in the Design Center resource set. It will created if an import is made.

The resource in he local Design Center resource set has the same version as the resource in the Document Services Repository. No need to import.

The local resource (in the Design Center resource set) has a newer version than the resource in the Document Services Repository. An import will overwrite the local resource with an older version.

The resource has been removed from the Document Services Repository, but still exists locally.

Scripting

In this chapter

Get the value of a SOM expression to a StreamServe variable

You can map the values from a SOM expression to a StreamServe variable.

This mapping makes it possible to use data from ADEP Designer. For example, to control post-processing or set driver options depending on input, template, form, layout or script data.

This mapping is one-way, i.e. you can get values from a SOM expression to a StreamServe variable, but not the other way around.

Mapping can be done once for each document, or once for each page:

- Per document – mapping is done once for each document, directly after all pages are produced, but before the document was output.
- Per page – mapping is done once before each page is output. This enables the use of more advanced functionality in Document Broker and sheet layout. For example, you can use a page-level value extracted from a SOM expression to create proper OMR codes in AFP output.

  The mapping is done first for the entire document, and then for each page in turn before it is output.

  Note: To use per page mapping, relative SOM expressions are used to refer to layout objects. Relative SOM expressions are evaluated in the page context, making it possible to have different values for different pages. Objects that are not in page context will be evaluated in document context, and have the same values for all pages.

Example 6 *SOM expression and StreamServe variable*

SOM expression: `$record.header.txtPONum`.

StreamServe variable: `$ponum`

Example 7 *Assigning a value to a StreamServe variable*

Input data to the XFA processor:

```
<?xml version="1.0" encoding="UTF-8"?>
<batch>
<transaction>
 <header>
  <txtPONum>1234567890</txtPONum>
  <dtmDate>2004-02-08</dtmDate>
  <txtOrderedByCompanyName>Another Company</
txtOrderedByCompanyName>
  <txtOrderedByAddress>123, Any St.</txtOrderedByAddress>
  …
```

The variable mapping will assign the value of the `txtPONum` element to the `$ponum` StreamServe variable:

SOM Expression: `$record.header.txtPONum`

StreamServe variable: `$ponum`

Example 8 *Mapping per page – relative SOM expression for master page children*

Relative SOM expressions for master page children should start from first level child of master page.

A master page, `Page1`, contains:

- text field `PageNumber`
- subform `CustomerInfo` with the fields `CustomerName` and `CustomerID`

Relative SOM expression for field `PageNumber` is "`PageNumber`".

Relative SOM expression for `CustomerName` is "`CustomerInfo.CustomerName`".

Example 9 *Mapping per page – relative SOM expressions for main subform children*

Relative SOM expressions for main subform children should start from first level (page level) child of main subform.

A main subform `Data` can be referred to by using the absolute SOM expression "`$form.Data`".

The subform `Data` contains the child subform `Part1`, which contains the field `Header1`.

Relative SOM expression for `Header1` is "`Part1.Header1`".

For information about the SOM expression syntax, see *Adobe Digital Enterprise Platform Document Services – Designer 10.0 Scripting Basics*.

About SOM expressions

The XFA Scripting Object Model (SOM) is a model for referencing values, properties and methods within a particular Document Object Model (DOM). A DOM structures objects and properties as a tree hierarchy. XFA SOM expressions provide easy access to these objects and properties through a straightforward object reference syntax.

The SOM specification is described in detail in the XML Forms Architecture (XFA) Specification.

To find out what the SOM expression to a particular XFA object is, use the `somExpression` scripting property. By using this, the full expression to the object can be extracted.

Example 10 *ECMA-script example*

```
this.rawValue = this.somExperssion;
```

If you place this script on the object for which you want the SOM expression, you will get the SOM expression assigned to the value of that object and the string will be printed in the output document.

For more information about SOM expressions, see *Adobe Digital Enterprise Platform Document Services – Designer 10.0 Scripting Basics*.

Access StreamServe variables in the XFA processor

StreamServe scripts variables can be read directly from a script in the XFA processor. The XFA processor adds the variables to a data set named `strs` which can be accessed from both Java script and FormCalc scripts in the loaded XDP template

All StreamServe variables created prior to the execution of the process can be accessed through the data connection by using the following syntax:

```
xfa.datasets.strs.variables.variableid.value
```

When a StreamServe Message is used as the data connection for a template (i.e. the setting **Add the Message as a data connection in the Data view** is enabled), the following syntax is also valid:

```
$record.variables.variableid;
```

Example 11 *StreamServe Script before process*

```
$myvar = "myvalue";
//Assigns "myvalue" to a StreamServe variable named "myvar"
```

Example 12 *ECMA-script on Form Ready on a field in template*

```
this.rawValue = xfa.datasets.strs.variables.myvar.value;
//Assigns the value of the StreamServe variable "myvar" to the raw
value of the current object.
```

Note: Read Only variables, such as those created by SAP (for example, RDI header variables) and Lawson agents, can not be read in this way. Such variable values must be assigned to scripting (Read/Write) variables before they can be read in the XFA processor.
The placement of the script and when the script is executed determines if you can use this method. The `pageSet` object and its descendants can safely be assigned to scripting variables only in the `layout:ready` event and the `prePRint` event.

Using Before and After Process scripts

You can run Before and After Process scripts, before and after each record.

Running scripts before and/or after individual records gives a script context for variable values, extracted using SOM expressions. See *StreamServe variable mapping* on page 88.

Mappings from SOM expressions to StreamServe variables are performed for each record (and even, if **Evaluate variable for each page** is enabled, for each page within a record). With this setting:

- Enabled – The Before/After Process script is invoked for each record with these values.
- Disabled – Only the values extracted from the last record will be available to the After Process script.

Note: The very first Before Process script and the very last After Process script will be executed in the pre-process phase (before and after the actual process) as well as in the process phase. All the other Before and After Process scripts will be executed only in the process phase.

How it works – script execution order

The execution order of Before and After Process scripts during the process phase, with the setting **Evaluate variable for each page** enabled, is illustrated in the figure below.

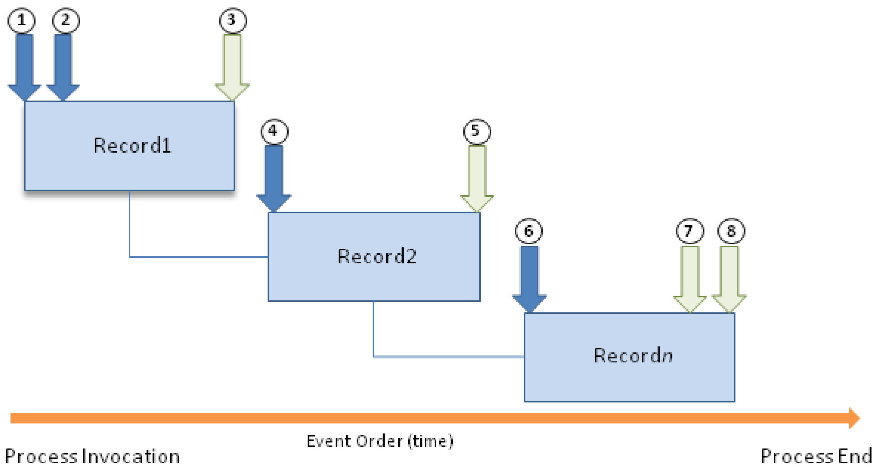


Figure 15 Script execution order

1 The Before Process script is executed before the process invocation.

2 The Before Process script is executed before the first record.

3 The After Process script is executed after the first record.

4 The Before Process script is executed before any subsequent record.

5 The After Process script is executed after any subsequent record.

6 The Before Process script is executed before the last record.

7 The After Process script is executed after the last record.

8 The After Process script is executed after the Process end.

To enable Before and After Process scripts

You enable and disable this functionality in the Settings dialog by selecting the **Run Before and After Process scripts, before and after each record**. See *Settings dialog box* on page 86.

If this setting is selected, Before and After Process scripts will be executed before and after each record. If not selected, the before and after Process scripts will be executed before and after each Process.

For new Projects created in Production Print, this functionality is enabled by default.

For upgraded Projects, created in versions before LiveCycle Production Print ES2, this functionality is disabled by default and the previous behavior with scripts per Process applies.

Restrictions

The following scripts cannot be used in the before and after Process scripts when running on record level in the process phase:

- `CallProc`
- `CallBlock`
- `PreProcLog`
- `EndDocument`
- `SetExtJobId`
- `GetJobResourceIndex`
- `AddSortDef`
- `AddSortKeys`
- `ODBCConnect`
- `ODBCDisconnect`
- `SAPCreateFunction`
- `SAPInvokeFunction2`
- `SetDestPath`

Logging to the StreamServer log file

You can write log messages from scripts in an XDP template directly to the StreamServer log.

Syntax

```
xfa.log.message(Param1, str [, severity])
```

`Param1`	This parameter is always ignored by StreamServer.
`str`	Message text string that will be written in the log fie. The message will be preceded by: "`XFAOUT: Message from XFA template script:` "
`Severity`	Optional. Specifies the severity of the message: `i` – Information `t` – Information `w` – Warning `f` – Error

Description

This script function makes it possible to write messages directly to the StreamServer from scripts in an XDP template.

Example

Example 13 *Log call example*

```
xfa.log.message(1, "text to write in log file", "i");
```

This will result in the following message in the StreamServe log:

```
0902 175411 (0096) 3 XFAOUT: Message from XFA template script: text
to write in log file
```

XFA support

The Production Print processor implements a subset of the XML Forms Architecture (XFA) 3.3 specification as listed in this section.

Functionality related to the creation of interactive document output is generally not supported, except for the drawing properties of such objects when statically rendered in print formats.

The same applies for the scripting implementation in FormCalc and ECMA-script.

Note: Production Print is developed for high performance processing of XFA templates. The XFA processor does not function exactly as the corresponding XFA processors in Document Services Output and Document Services Forms. For performance reasons, there are differences in how script contexts can be used as described in this section.

In this chapter

XFA elements

In this chapter

Fully supported XFA elements

The following XFA elements are fully supported. See Adobe XML Forms Architecture (XFA) for the latest XFA specification.

- `<arc>`
- `<area>`
- `<bind>`
- `<boolean>`
- `<checkButton>`
- `<color>`
- `<comb>`
- `<contentArea>`
- `<corner>`
- `<date>`
- `<dateTime>`
- `<defaultUi>`
- `<draw>`
- `<edge>`
- `<extras>`
- `<fill>`
- `<float>`
- `<font>`
- `<format>`
- `<image>`
- `<integer>`
- `<line>`
- `<linear>`
- `<margin>`
- `<occur>`
- `<overflow>`
- `<pattern>`
- `<picture>`
- `<proto>`
- `<radial>`
- `<rectangle>`
- `<solid>`
- `<stipple>`
- `<subformSet>`
- `<time>`
- `<ui>`
- `<variables>`
- `<validate>`

- `<hyphenation>`

XFA elements with unsupported attributes

The following XFA elements are partially supported. See the *Template Reference* section in the Adobe XML Forms Architecture (XFA) for detailed information about the XFA elements.

Element	Unsupported attributes
`<barcode>`	• `charEncoding` • `dataPrep` • `printCheckDigit` • `rowColumnRatio` • `upsMode`
`<border>`	• `break`
`<break>`	• `bookendLeader` • `bookendTrailer` • `after` • `before`
`<breakAfter>`	• `leader` • `trailer` • `targetType`
`<breakBefore>`	• `leader` • `trailer` • `targetType`
`<button>`	• `highlight`
`<calculate>`	• `override`
`<caption>`	• `placement`
`<choiceList>`	• `commitOn` • `open` • `textEntry`
`<dateTimeEdit>`	• `hScrollPolicy` • `picker`
`<decimal>`	• `leadDigits`

Element	Unsupported attributes
`<event>`	• `activity`
`<exData>`	• `maxLength` • `rid` • `transferEncoding`
`<exclGroup>`	• `access` • `accessKey`
`<field>`	• `access` • `accessKey`
`<imageEdit>`	• `data`
`<items>`	• `ref`
`<keep>`	• `intact` • `next` • `previous`
`<medium>`	• `stock` • `trayIn` • `trayOut`
`<numericEdit>`	• `hScrollPolicy`
`<pageArea>`	• `blankOrNotBlank` • `initialNumber` • `numbered` • `oddOrEven` • `pagePosition`
`<pageSet>`	• `relation`
`<para>`	• `preserve`
`<passwordEdit>`	• `hScrollPolicy`
`<script>`	• `runAt`
`<setProperty>`	• `connection`
`<subform>`	• `access` • `allowMacro` • `layout` • `restoreState`

Element	Unsupported attributes
`<template>`	• `baseProfile`
`<text>`	• `rid`
`<textEdit>`	• `allowRichText` • `hScrollPolicy` • `vScrollPolicy`
`<value>`	• `override`

XFA elements used only to add data to tagged PDF

The following XFA elements are only used to add data to tagged PDF. See the *Template Reference* section in the XFA specification, see Adobe XML Forms Architecture (XFA) for detailed information.

- `<assist>`
- `<desc>`
- `<speak>`
- `<toolTip>`
- `<traversal>`
- `<traverse>`

XFA element used only to embed flash objects

Flash objects can be embedded in an XFA template. Even though flash integration is not relevant for a Production Print solution, Production Print can still recognize a flash object, render it as inactive, and display it as a poster image. The output is visually identical to the corresponding output from Document Services Output, but without the ability to activate the actual flash content.

Production Print identifies flash content from the `classId` attribute in an `<exObject>` element. Production Print supports the `<exObject>` element only when it is used to embed flash objects. All `<exObject>` children except the `<image>` child with the `name` attribute `"poster"` will be ignored.

You can script against all `<exObject>` children, but only scripting against the `<image>` child with the `name` attribute `"poster"` will have any visible effect. The `exObject.setState` and `exObject.invoke` methods are ignored.

For detailed about embedding flash objects, see the Adobe user documentation.

XFA bookmarks

Production Print supports generation of bookmarks according to the XFA 3.3 specification with the following exceptions:

- The `<bookmarkGenerationPolicy>` options `server` and `client` both results in the `server` option when the bookmarks are generated.
- When clicking a bookmark in the pane, the corresponding page is always displayed. Any `setFocus` action types will be handled like the `gotoPage` action type and any `runScript` action types will be ignored.

Note that Production Print only supports bookmarks in PDF documents, created via the StreamServe PDF driver.

For detailed about XFA bookmarks, see the Adobe User Documentation.

Not supported XFA elements

The following XFA elements are not supported.

- `<appearanceFilter>`
- `<certificates>`
- `<digestMethod>`
- `<encoding>`
- `<encrypt>`
- `<execute>`
- `<handler>`
- `<keyUsage>`
- `<manifest>`
- `<message>`
- `<oids>`
- `<reasons>`
- `<signData>`
- `<signing>`
- `<subjectDNs>`
- `<timeStamp>`
- `<bindItems>`
- `<certificate>`
- `<connect>`
- `<digestMethods>`
- `<encodings>`
- `<filter>`
- `<issuers>`
- `<lockDocument>`
- `<mdp>`
- `<oid>`
- `<reason>`
- `<ref>`
- `<signature>`
- `<subjectDN>`
- `<submit>`

XFA scripting

In this section

Supported script functions

The script functions in the table below have full or limited support. If a function is not found in the table, it is not implemented.

- Fully supported functions work according to the *Adobe Digital Enterprise Platform Document Services – Designer 10.0 Scripting Reference*.
- Partially supported functions work according to the *Designer Scripting Reference* with the exceptions shown in the table below.

Function	**Comments**
`absPage`	Full support
`absPageCount`	Full support
`absPageSpan`	Full support
`addInstance`	• Boolean parameter `param` is not supported. • The new instance is not merged with the data DOM. • This function will fail if the `maxOccurs` attribute of a subform within its given context is exceeded by adding the subform.
`addItem`	Full support
`assignNode`	• This function only works for nodes in the form DOM or in the layout DOM. • The `param3` parameter is not supported. • Creation of new nodes does not work. • This function will always try to assign the value to an existing node, or else it fails.
`boundItem`	Full support
`clearItems`	Full support

Function	Comments
`clone`	This function only works for nodes in the form DOM or in the layout DOM.
`deleteItem`	Full support
`execCalculate`	This function is only supported for `exclGroup`, `field`, `form` and `subform`.
`execEvent`	Full support
`execInitialize`	This function is only supported for `exclGroup`, `field`, `form` and `subform`.
`getAttribute`	Full support
`getDisplayItem`	Full support
`getElement`	Full support
`getItemState`	Full support
`getSaveItem`	Full support
`h`	• This function is only supported for `field`, `draw`, `subform` and `contentArea`. • The `param3` parameter must be 0 otherwise this function will return 0.

`isCompatibleNS`	• For objects in the template DOM, the form DOM and the layout DOM, this function will return true for the following namespaces:
	`http://www.xfa.org/schema/xfa-template/`
	`http://www.xfa.org/schema/xfa-template/2.1/`
	`http://www.xfa.org/schema/xfa-template/2.4/`
	`http://www.xfa.org/schema/xfa-template/2.5/`
	`http://www.xfa.org/schema/xfa-template/2.6/`
	• For objects in the data DOM, this function will return true for:
	`http://www.xfa.org/schema/xfa-data/`
	• All other namespaces-object model combinations will return false.

`isPropertySpecified`	Full support
`item`	Full support
`message`	Full support
`messageBox`	• Implemented in java script only, but no message box is displayed. • Method always returns 2 (cancel).

Function	Comments
`namedItem`	Full support
`page`	Full support
`pageContent`	• No support for getting all page content (i.e. specifying "empty" for `param2`). • No support for `param3`. • This function will always search the entire page including both `pageArea` and `contentAreas`.
`pageCount`	Full support
`pageSpan`	Full support
`relayout`	• Limited functionality. The current layout is completely finished, and then a second layout pass will run. This second pass will recreate all pages, and layout all objects from the start on the pages. • Scripts on form nodes are not rerun after this re-layout. Only scripts present on overflow headers and footers and boilerplate objects will run.
`remerge`	This function will do a complete remake of the form DOM, but this remake will only be done once. If `remerge` is called again in the second pass of merging the DOM, it will have no effect.
`removeInstance`	Full support
`resolveNode`	Full support
`resolveNodes`	Full support
`saveXML`	This function is implemented, but the returned XML cannot be guaranteed to be correct.
`selectedMember`	Full support
`setAttribute`	Full support
`setElement`	This function is implemented, but does not work correctly. Replacing elements will not always be reflected in the output.
`setItems`	Full support
`setItemState`	Full support
`sheet`	This function always returns the same value as `page`.
`sheetCount`	This function always returns the same value as `page`.

Function	**Comments**
`w`	• This function is only supported for `field`, `draw`, `subform`, `contentArea` and `pageArea`. • The `param3` parameter must be 0 otherwise this function will return 0.
`x`	The `param3` parameter must be 0 otherwise this function will return 0.
`y`	The `param3` parameter must be 0 otherwise this function will return 0.

Supported script properties

The script properties in the table below have full or limited support. If a property is not found in the table, it is not implemented.

- Fully supported properties work according to the *Adobe Digital Enterprise Platform Document Services – Designer 10.0 Scripting Reference*.
- Partially supported properties work according to the *Designer Scripting Reference* with the exceptions shown in the table below.

Property	**Comments**
`#text`	Full support
`{default}`	Full support
`activity`	This property is read only, and cannot be set from scripts.
`after`	Full support
`afterTarget`	When setting this property, only names of existing subforms in the current template can be used.
`all`	Full support
`allowNeutral`	Full support
`allowRichText`	Full support
`anchorType`	Full support
`aspect`	Full support
`baselineShift`	Full support
`before`	Full support
`beforeTarget`	When setting this property, only names of existing subforms in the current template can be used.

Property	Comments
`borderColor`	Full support
`borderWidth`	Full support
`bottomInset`	Full support
`break`	Full support
`cap`	Full support
`charEncoding`	Full support
`checksum`	Full support
`circular`	Full support
`classAll`	Full support
`classIndex`	Full support
`className`	Full support
`colSpan`	Full support
`columnWidths`	Full support
`contentType`	This property is supported for the following objects: • `exData` (read/write) • `image` (read/write) • `script` (read only).
`count`	Full support
`cSpace`	Full support
`data`	Full support
`dataColumnCount`	Full support
`dataLength`	Full support
`dataRowCount`	Full support
`editValue`	This property sets or gets the `rawValue`. The distinction between `rawValue` and `editValue` is not done.
`endChar`	Full support
`errorCorrectionLevel`	Full support
`excludeAllCaps`	Hyphenation implementation differs between ADEP and StreamServe, so there is no guarantee that hyphenated text will look the same.

Property	**Comments**
`excludeInitialCap`	Hyphenation implementation differs between ADEP and StreamServe, so there is no guarantee that hyphenated text will look the same.
`fillColor`	Full support
`fontColor`	Full support
`fontHorizontalScale`	Full support
`fontVerticalScale`	Full support
`formattedValue`	Setting this property will set the `rawValue` property. Getting this property will apply formatting to the `rawValue` before returning to the caller.
`fracDigits`	Full support
`h`	Full support
`hAlign`	Full support
`hand`	Full support
`highlight`	Full support
`href`	Only used in `image` objects. Can be set and read in `exData` as well, but it has no meaning in that object.
`id`	Read only
`index`	Read only
`initial`	Full support
`instanceIndex`	Read only
`intact`	Full support
`inverted`	Full support
`isContainer`	Full support
`isNull`	Full support
`join`	Full support
`kerningMode`	Full support
`ladderCount`	Full support
`layout`	Full support
`leadDigits`	Full support
`leader`	Supported for `breakAfter`, `breakBefore` and `overflow`.

Property	Comments
`leftInset`	Full support
`length`	Full support
`letterSpacing`	Full support
`lineHeight`	Full support
`lineThrough`	Full support
`lineThroughPeriod`	Full support
`listen`	Read only
`locale`	Full support
`long`	Full support
`marginLeft`	Full support
`marginRight`	Full support
`mark`	Full support
`bind`	Read only
`max`	Full support
`maxChars`	Full support
`maxH`	Full support
`maxW`	Full support
`min`	Full support
`minH`	Full support
`minW`	Full support
`moduleHeight`	Full support
`moduleWidth`	Full support
`multiLine`	Full support
`name`	Full support
`next`	The `pageArea` option is not supported.
`nodes`	Full support
`ns`	Read only
`numberOfCells`	Full support
`numPages`	Read only

Property	Comments
`orientation`	Full support
`overflowLeader`	Read only
`overflowTarget`	Read only
`overflowTrailer`	Read only
`override`	Read only. Can be set but it makes no difference once the template is loaded.
`parent`	Read only
`parentSubform`	Read only
`passwordChar`	Full support
`placement`	Full support
`posture`	Full support
`presence`	Full support
`preserve`	Full support
`previous`	The `pageArea` option is not supported.
`printCheckDigit`	Full support
`pushCharacterCount`	Hyphenation implementation differs between ADEP and StreamServe, so there is no guarantee that hyphenated text will look the same.
`radius`	Full support
`radixOffset`	Full support
`rate`	Full support
`rawValue`	`formattedValue` and `editValue` are the same values as the `rawValue` in Production Print. Setting either of these values will result in all of them being set to the same value. `formattedValue` will however be formatted once more when output. This is a known issue.
`ref`	Read only
`relation`	Read only
`relevant`	Full support
`remainCharacterCount`	Hyphenation implementation differs between ADEP and StreamServe, so there is no guarantee that hyphenated text will look the same.

Property	**Comments**
`reserve`	Full support
`rightInset`	Full support
`rotate`	Full support
`runAt`	Treat as read only. Can be set but will not have any effect.
`save`	Full support
`scope`	Read only.
`selectedIndex`	Full support
`shape`	Full support
`short`	Full support
`size`	Full support
`slope`	Full support
`somExpression`	Read only
`spaceAbove`	Full support
`spaceBelow`	Full support
`startAngle`	Full support
`startChar`	Full support
`startNew`	Full support
`stroke`	Full support
`sweepAngle`	Full support
`tabDefault`	Full support
`target`	Supported for `breakAfter` and `breakBefore`. When setting this property, only names of existing subforms in the current template can be used.
`targetType`	No support for options `pageEven` and `pageOdd`.
`textIndent`	Full support
`textLocation`	Full support
`thickness`	Full support
`this`	Full support
`topInset`	Full support

Property	**Comments**
`trailer`	Supported for `breakAfter` and `breakBefore`. When setting this property then only names of existing subforms in the current template can be used.
`transferEncoding`	Supported for `image` only.
`truncate`	Full support
`type`	Supported for `barcode`, `linear`, `pattern` and `radial`.
`typeface`	Supported, but the corresponding font must be included in the StreamServe Project.
`underline`	Full support
`underlinePeriod`	Full support
`use`	Read only
`usehref`	Read only
`vAlign`	Full support
`value`	For objects that support `rawValue`, value and `rawValue` will return the same result and set the same value.
`w`	Full support
`weight`	Full support
`wideNarrowRatio`	Full support
`wordCharacterCount`	Hyphenation implementation differs between ADEP and StreamServe, so there is no guarantee that hyphenated text will look the same.
`x`	Full support
`y`	Full support

Supported script object models

The script object models in the list below have full or limited support. For information on limited support, see:

- *Supported script functions* on page 115
- *Supported script properties* on page 118

If an object model is not found in the list, it is not supported.

- `Data model`
- `Form model`
- `Host model`
- `Layout model`
- `Log model`
- `XFA model`

Supported events

The form events in the list below have full support. Fully supported events work according to *Adobe Digital Enterprise Platform Document Services – Designer 10.0 Scripting Basics.*

If an event is not found in the list, it is not implemented.

- `calculate`
- `doc:ready`

  Note: You must manually enable execution of the `doc:ready` event. See *Enabling execution of the doc:ready event* on page 125.

- `form:ready`
- `indexChange`
- `initialize`
- `layout:ready`
- `prePrint`
- `validate`

Enabling execution of the doc:ready event

The `doc:ready` event is disabled by default. To enable execution of the event, you must add a command to the Edit Custom Fields dialog box in Design Center.

If you want to disable the `doc:ready` event later, you can simply remove the command.

To enable execution of the doc:ready event

1. In Design Center, select the Runtime view.

2 Select **Edit** > **Custom Settings**. The Edit Custom Fields dialog box opens.
3 In the **Access points** browser, browse to and select the logical node for the ADEP Designer Process.
4 On the **Custom** sheet, enter the following command:

```
EnableDocReadyEvent 1
```

5 Click **OK**.

Considerations when scripting

Scripting in the layout DOM

The layout DOM is first created completely, and distributed over pages, content areas, etc. After this, all layout-ready and preprint scripts are run in the order specified in the XFA 3.3 specification. If any layout-ready or preprint script changes the layout DOM in a way that may cause objects to switch page, the layout DOM will be completely rebuilt.

Rebuilding the layout DOM will delete:

- all dynamically created layout objects, such as overflow leaders and overflow trailers.
- all objects on master pages.

All other layout objects (such as fields from the form DOM) are preserved, and so are the values of these layout objects.

To preserve the correct values, no layout scripts for form objects are run after the layout DOM is rebuilt. Only scripts attached to the dynamically created layout objects (overflow leaders, master page objects etc.) are run this time.

The layout DOM scripts will only run once for each object with some exceptions:

- The layout-ready and preprint scripts on the root subform will be run both the first and the second time (if any) that the layout DOM is created and arranged.
- Initialize and calculate scripts for master page objects can be executed twice.

This means you must be careful when using global variables and when initializing them, or when accessing fields and other values in dynamically created subforms in objects in the form DOM:

- Any initialization of global variables, and any modifications done externally (from other parts of the DOM) to dynamically created layout objects, must be done in the root subform or in script functions that are called from the root subform.
- Updating values in form DOM objects from within e.g. an overflow header can cause unexpected results.

Example: `subform.field.x = subform.field.x + 100;` in an overflow leader will cause the `subform.field.x` to be increased by 100 the first time the layout DOM is created. If the layout DOM is recreated once more, 100 more will be added to `x`.

Global variables declared inline in java scripts

- Global variables declared inline in java scripts in the form DOM, e.g. `initialize` or `form ready`, cannot be used in the layout DOM with the initialized values. There is one exception to this – global variables declared in dedicated script objects. These variables can be initialized in form DOM scripts and be used in layout DOM scripts.
- Global variables declared inline in java scripts cannot be used in form calc scripts and SOM expressions (for example to transfer values to StreamServe variables). To be able to use these variables, they must be declared as form variables (i.e. they must be visible in the hierarchy tree in ADEP Designer).

Script execution order

The script execution order for objects within an Event type in XFA is undefined. Do not write scripts that are depending on any particular script execution order.

Glossary

Some terms and concepts used in this document.

ADEP	Adobe® Digital Enterprise Platform
ADEP Designer	The Adobe forms design tool.
ADEP Designer Process tool	The application, developed by OpenText, that integrates the ADEP Designer in StreamServe Design Center where it is used as a StreamServe Process tool.
Control Center	StreamServe Control Center is used to manage and administer StreamServe applications.
Design Center	The StreamServe configuration tool for your StreamServe Projects.
Dependencies	Fragments and images referenced from within an XDP file – the main XDP file.
Deploy a Project	When you deploy a Project, the export file is used to unpack and store the configuration files in the working directory of the StreamServer application.
Document	A Document (capital D) is a grouping of documents per customer number, delivery address, etc. The scope of the Document is specified using a Document trigger.
Event	An Event controls how to identify and extract fields from the input data, and how to structure and label the extracted fields.
Export a Project	When you export a Project, you generate a package file (`*.export`) that contains all the configuration files needed to run a Project. You export from Design Center.
LCA	LiveCycle Archive
Message	A Message manages conversion from input to output format. A Project normally contains several Messages, where each Message corresponds to a specific document type. For example, one Message for invoices, another for orders, etc. A Message contains Events and Processes.

PageIN	The Event tool that handles page formatted input data. For other Event tools, see *StreamServe Persuasion SP5 Design Center* documentation.
PageOUT	The Process tool that produces page oriented output data. For other Process tools, see *StreamServe Persuasion SP5 Design Center* documentation.
Platform	The Platform is where you configure the environment settings for a Project. For example, in the Platform you specify how to connect to and receive input from source applications, and how to connect to and deliver output to the output devices (printers, faxes, etc.).
Process	A Process controls which fields to retrieve, and how to structure the fields in the output delivered from the StreamServer.
Project	A StreamServe Project is a definition of how a specific EDP solution should collect, transform and deliver data.
Resource	A resource in is a file with an embedded source file. For example, to be able to use the image file `logo.gif` in the Project, this file must be embedded in a resource. All external files, except dependencies, that you refer to when you configure a Project must be converted to resources.
Resource set	A set of links to resource files.
Runtime	The Runtime configuration is where you connect Messages to the Platform. A Project normally contains several Runtime configurations, for example one Runtime configuration per Message.
StreamIN	The Event tool that handles field- or record-based input data.
StreamServer	The component that handles the collection and transformation of input data, and the delivery of output documents. The StreamServer can run several StreamServer applications, where each StreamServer application is dedicated to a specific StreamServe Project.
XDP	XML Data Package.
XFA	XML Forms Architecture

XFA processor	The application, developed by StreamServe, that enables the StreamServer runtime environment to generate output in various formats from XML Data files and ADEP Designer form templates.
XMLIN	The Event tool that handles XML formatted input data.