



**pdfEXPRESS™**  
**Scripting Reference**

**Version 1.0.5 Plug-in Build 1640 and later, 12/5/2000**

Copyright © 2000 by Think121.com.  
All Rights Reserved.



# Table of Contents

|                             |    |
|-----------------------------|----|
| 1.0 Introduction.....       | 5  |
| 2.0 Scripting Syntax .....  | 7  |
| 3.0 pdfExpress Syntax ..... | 9  |
| 4.0 Merge Engine.....       | 17 |



# 1.0 Introduction

pdfExpress scripting is the process whereby a user extracts information from a database, web server, spreadsheet, or other data based application and, using that information, uses pdfExpress to assemble or modify one or more PDF files. pdfExpress does not, like some applications, provide a direct interface to the database. Instead, pdfExpress assumes that some other application has extracted the data and prepared it for pdfExpress.

pdfExpress is designed to be fully scriptable either programmatically (via an AppleScript or a COM interface) or via a script file. This document defines the scripting syntax and semantics. The commands defined are usable in all versions of pdfExpress.

Scripting has two uses within pdfExpress. The first is to control the pdfExpress merge engine. During the merge process pdfExpress interacts with PDF files and data to create new composite PDF files. The second use of scripting with pdfExpress is to create an extended user interface for Acrobat. In this application, both the pdfExpress Acrobat plug in and an external application exchange information in real time via the Scripting language. Contact think121.com for more information on this use of pdfExpress.

A support library (available as a COM or AppleScript add-on) is available for general use to decode and manipulate pdfExpress scripts. This support library allows the user to parse and decompose pdfExpress scripts directly from within a variety of programming environments. This library is particularly useful when writing real-time data exchange applications to extend the Acrobat user interface.



## 2.0

# Scripting Syntax

The pdfExpress scripting syntax interface supports both a PostScript-style syntax as well as an XML-based syntax.

All pdfExpress scripting is declarative in nature and does not involve programming or flow-control constructs. Generally scripts are created by third-party applications such as databases, web servers, spreadsheets and so forth.

pdfExpress scripts are written in one of the two formats (PostScript or XML) as mentioned above. The actual pdfExpress commands (as opposed to the syntax by which the commands are defined) are defined within this chapter. The PostScript-style syntax for pdfExpress commands follows syntax of PDF as described in Section 4 of the Adobe Portable Document Reference Format Reference Manual Version 1.3 (March 11, 1999). For reference this document should be downloaded from the Adobe web site at [www.adobe.com](http://www.adobe.com). Users familiar with PostScript will be able to start using this syntax format immediately.

pdfExpress commands can also be described via an XML using the pdfExpress DTD. This DTD, available from think121.com, defines XML tags that offer the same syntactical capabilities as the PostScript-style syntax. This means that any pdfExpress script can be expressed identically in either format.

A pdfExpress script file is processed by sequentially scanning each contained pdfExpress command from the beginning of the file to the end. Each pdfExpress command is defined by a single array followed by a **merge** command. Although pdfExpress syntax follows the syntax of PDF, the pdfExpress commands themselves (as well as the **merge** command) are not part of PDF; only the syntax (or structure) of the commands is the same.

pdfExpress commands expressed in the PDF/PostScript form have the property that they can be embedded with a PDF file. This allows applications to create pdfExpress scripts

and pass them around as part of PDF files.

The script file is consumed by the merge engine command-by-command until end of file is reached. If no errors occur by the time end of file is reached, the script is presumed to have completed successfully. If either the XML or PostScript parser encounters a syntax error during the processing of the file, an error is reported to the user (either interactively or via a log file). Please note that sometimes errors can occur in script files (particularly those written with the PostScript style) which pdfExpress can easily detect and recover from. For example, an unterminated array or string. In these cases the error message provided may be less than helpful. To a lesser extent the same is true of XML files, though the pdfExpress XML parser attempts to locate the line number and character position of all such errors.

If you should encounter such a non-obvious error, the most effective way of discovering it is to simplify the script file bit-by-bit until a working version is obtained. For example, first removing all but one /addPage, next by simplifying the substitution dictionary, and so on until the file begins to work.

Elements of PDF syntax used by pdfExpress include comments, boolean, numbers, integer, strings, text, names, arrays and dictionaries. PDF-style streams, objects, nulls, and indirect references are not used by pdfExpress. For convenience the following sections briefly describe the syntax of the elements used.

Please note that all pdfExpress (as well as PDF and PostScript) language elements are case sensitive, i.e., Abc is considered different from abc.

# 3.0 pdfExpress Syntax

pdfExpress syntax is defined in terms of both PDF and XML. pdfExpress defines an XML construct to represent each of the corresponding PDF constructs, e.g., comments, booleans, and so forth.

The representations are designed to make it possible to create automated tools to convert between the two formats. Both syntactical representations are provided.

The following syntax conventions are used:

|       |          |   |
|-------|----------|---|
| ...   | ellipsis | indicate required information                           |
| 'x'   | quotes   | indicates that the character 'x' is required literally. |
|       | or       | separates two or more choices on a line                 |
| [ x ] | option   | x is optional, i.e., x or xy is valid                   |
| x*    | zrepeat  | x is repeated zero or more times                        |
| x+    | orepeat  | x is repeated one or more times                         |
| { }   | group    | items enclosed in {}'s are grouped                      |

## 3.1 Comments

Comments are used to add information to a pdfExpress scripting file. The information contained by the comments does not affect the operation of pdfExpress in any way.

**Acrobat Syntax:** ‘%’...<end of line>

**XML Syntax:** ‘<!--’ ... **comment characters**... ‘—>’

In the native PostScript-style form pdfExpress comments are introduced by the percent sign (%). Text between the first percent sign (outside of a String as defined below) and the end of the line is ignored. For example

```
[ /begin ] merge
  %% This text is ignored.
[ /end ] merge
```

The text following the first ‘%’ is ignored until the end of line is reached.

Comments may also occur within pdfExpress commands. For example

```
[ /begin
  % This is ignored.
] merge
```

These three lines comprise a single merge command. The comment after the **/begin** is ignored through the end of the line on which it appears.

XML comments follow the XML comment convention and may appear anywhere legally allowed by the XML Standard. For example

```
<!--
  Canonical pdfExpress sample file
-->
<pdfXMergeScript>
...
</pdfXMergeScript>
```

demonstrates an XML comment that appears before the outermost tags. Note that XML comments cannot appear inside of XML tags.

## 3.2 Data Types

pdfExpress scripting is defined in terms of six basic data types: boolean, number, string, name, array and dictionary. All pdfExpress commands are made up of one or more of these data types. Although outlined in much more detail in the previously mentioned Adobe documentation, a brief summary of their form and syntax is provided below:

### 3.2.1 Boolean

Boolean values provide a means of communicating yes and no values to pdfExpress. For example, enabling certain features within the product.

**Acrobat Syntax:** True | False  
**XML Syntax:** <boolean value="true" />  
<boolean value="false" />

Unlike the rest of the pdfExpress syntax the symbols for true and false are case insensitive. Boolean values are typically used as part of dictionaries. For example

```
[ /begin << /AllPagesSupportMediaBox true >> ] merge
```

The dictionary associated with the **/begin** command assigns the value **true** to the AllPagesSupportMediaBox attribute.

### 3.2.2 Numbers

**Acrobat Syntax:** [+|-]digit+[.[digit+[e|E[+|-]digit+]] ]  
**XML Syntax:** <integer value="123" />  
<fixed value="123.456" />

pdfExpress numbers are represented by standard PDF syntax. The representation of pdfExpress numbers is either a 32-bit fixed-point number or a simple integer. Whole number values within the range of +/- 2,147,483,648 are treated as integers. All other numbers are treated as fixed point values.

For XML numbers, the text within the **value** attribute must be in the Acrobat number format.

The range for a non-integer number containing a decimal point is +/- 32,767.99996. The smallest fractional part of a decimal that can be resolved by pdfExpress is +/- .0000305. Numerically smaller values are rounded to the nearest .0000305<sup>th</sup> value.

The pdfExpress numeric representation must not be used for representing data values. Instead all numeric data values should be processed and transmitted as strings.

### 3.2.3 Strings and Text

Most pdfExpress values are represented as strings. String values are used to represent file names, paths, carousel names, and so forth.

**Acrobat Syntax:** ‘( { ASCII\_character | special\_character } \* ‘)’  
**XML Syntax:** <string escaped="true" value="&ab(c)" />  
<string value="&ab(c)" />  
<string escaped="true">  
    &amp;ab(c)  
</string>  
<string>&amp;ab(c)</string>

Unlike most computer languages and just like PostScript and PDF, Acrobat strings begin with an open-parenthesis and end with a close-parenthesis. Most printing characters (excluding ‘(, ’) and ‘\’) may be included between the opening and closing parenthesis. Non-printing characters may also be used, though the resulting files are harder to manipulate with standard tools.

Special characters, e.g., carriage return <CR>, are represented by standard backslash escape sequences, e.g., \015. Parenthesis and backslash characters within a string must always be escaped with backslash (\). This is particularly important in DOS and Windows file names.

For XML, pdfExpress provides two formats for strings. In the first, the value of the string is provided with the **value** attribute. In this case the entire value of the attribute is represented with value (note that XML strips leading and trailing spaces from all attributes). If the **escaped** attribute is present and true, then the XML string (this applies to both formats of **string**) is assumed to have the escape characters already embedded. If the **escaped** attribute is not present, then the string is assumed not to have escapes embedded.

The second string format for XML is similar to the first with the exception that the string's value is delimited by <string> and </string> rather than being defined as part of a attribute.

For example, the following are valid pdfExpress strings:

(\)) or, equivalently <string escaped=true value="\)" />  
(This is a string containing \ (parentheses\).) or, equivalently  
<string escaped="true" value="This is a string containing \ (parentheses\)." />  
<string value="This is a string containing (parentheses)." />  
<string escaped="true">This is a string containing \ (parentheses\).</string>  
(This string contains a carriage return ‘\015’ and a linefeed ‘\012’)  
(This string contains quotes “ ‘.)

An extensive discussion of string representation is located in the PDF Reference Manual 1.3, Section 4.4. Text as described in Section 4.4.1 is not supported.

### 3.2.4. Names and Tags

**Acrobat Syntax:** Name | /Tag  
**XML Syntax:** <name value="name" />  
 or  
 <tag value="tag">

The value of a name (for either a tag or name) is a sequence of characters which exclude the following: any white space (space, newline, carriage return, tab), (, ), [, ], <, >, {, }, %, and /.

### 3.2.5 Arrays

**Acrobat Syntax:** '[' array\_element\* ']'  
**XML Syntax:** <list>array\_element\*</list>

Arrays contain zero or more pdfExpress objects. Arrays are ordered and take their size from the number of elements between the square brackets. Array are introduced by an open square brace ([]) and terminated by a close square brace (]). For XML, an array is introduced with <list> and ends with </list>. Unlike dictionaries below, arrays are ordered with the first element considered to be the zeroth element. An array may have zero or more elements.

An array element may be another array, a boolean value, a number, a string, a name, array, or a dictionary.

### 3.2.6 Dictionaries

**Acrobat Syntax:** '<< { /TagName tagvalue }\* '>>'  
**XML Syntax:** <option>  
                   <key value="name">element</key>  
                   ...  
                   </option>

A dictionary is an object which allows named tags to act as keys and allows those keys to be associated with values. Each tag is a pdfExpress tag with the name of the tag being the actual tag value, i.e., /Fred is the tag "Fred". Each tag has a corresponding value which can be any pdfExpress object, e.g., boolean, list, dictionary, etc. Each tag that acts as a key in the dictionary must be unique.

A dictionary consists of zero or more name/value pairs. With Acrobat syntax each name/value pair is introduced by a slash (/) followed by a name. In XML, each entry is defined

by the <key> construct. The name is the key for the indicated value. A value may be any PDF element including an array, a boolean value, a number, a string, a name, or another dictionary. Each name in the current lexographic scope must be unique, i.e., within a given dictionary all names must appear only once unless they appear in a nested dictionary that is part of a value.

### 3.3 An Example

Following is a sample from an actual .LIS (PDF/PostScript-style) and an actual .LXS (XML) scripting file.

```
[ /begin <<
  /MergeMode /Server
  /AddPlatformSignature true
  /ExpandPathMacros true
  /MacroDefs <<
    /Letter (${_thisDir}letter.pdf)
    /Flags (${_thisDir}isigflag_g.pdf)
  >>
>> ] merge
```

In the above example, a typical pdfExpress **merge** command is defined. Note that the command itself is made up of a name (the word **merge**) and an array. The array contains two objects. The first is a tag (**/begin**). The second is a dictionary. The dictionary contains four entries. The first entry has the key **/MergeMode** and the value **/Server**. The second entry has the key **/AddPlatformSignature** and the value **true**. The third entry has the key **/ExpandPathMacros** and the value **true**. The fourth entry has the key **/MacroDefs** and the value is another dictionary. This dictionary is made up of two keys (**/Letter** and **/Flags**) that each have a string value.

The XML form of the same code is as follows:

```
<List>
  <Tag value="begin" />
  <Options>
    <Key value="MergeMode">
      <Tag value="Server" />
    </Key>
    <Key value="AddPlatformSignature">
      <Boolean value="true" />
    </Key>
    <Key value="ExpandPathMacros">
      <Boolean value="true" />
    </Key>
    <Key value="MacroDefs">
      <Options>
        <Key value="Letter">
          <String>${_thisDir}letter.pdf</String>
        </Key>
        <Key value="Flags">
          <String>${_thisDir}isigflag_g.pdf</String>
        </Key>
      </Options>
    </Key>
  </Options>
</List>
<Name value="merge" />
```

Note that in the above code the `${...}` constructs are part of string, not part of the scripting syntax. Also note that the XML format is exceptionally verbose.



# 4.0 Merge Engine

The pdfExpress merge engine is available as an Acrobat plug-in (both PC and Mac) and as a server. The syntax and commands supplied below are applicable to both versions. The only difference between the versions is support for file names: representation of a file name on each platform is particular to that platform.

pdfExpress can be queried to determine its version. This functionality is useful for scripts that are required to work with multiple versions of pdfExpress.

Certain limitations apply to the plug-in version of pdfExpress in a general computer environment. In particular, PDF and Acrobat were designed to operate in a desktop computer environment in which the Acrobat program itself does not need to multi-task. If multi-tasking is required, the server version should be used. Similarly, there are licensing restrictions on the use of the Acrobat program in multi-user environments; the pdfExpress server version is not limited by the Acrobat license issues.

Within the Adobe Acrobat environment, the pdfExpress scripting model allows the user to execute scripts while performing other tasks. However, certain functions cannot be shared among multiple, simultaneous users. To protect Acrobat and its execution environment external access to pdfExpress functionality is controlled via a merge engine interface. With the server version, a new instance of the executable should be started for each task. In either case, only one merge engine is active at any given time. This interface ensures that critical Acrobat resources are properly controlled and shared among multiple users.

(Due to the verbosity of XML, only the Acrobat versions of the commands are provided.)

## 4.1 Identifying the Version

**Acrobat Syntax:** [ /Identify ] merge

This command causes the engine to respond with its major version number via an **/Identity** response. The **/Identity** response has the following syntax: [ /Identity (*versionInfo*) ].

## 4.2 Initializing and Terminating

**Acrobat Syntax:** [ /begin ] merge  
[ /begin << ...options... >> ] merge

The pdfExpress merge engine is initialized by the **/begin** command. Only one merge engine is active at any given time and each engine must be terminated with the **/end** command before a new one can be started. An error will occur if an attempt to start more than one engine is encountered.

Options may be supplied with the begin command. All options have a default value which each new instance of the merge engine is initialized with (note that ending and restarting the merge engine returns these values to the defaults listed below).

Supported options include:

**/MergeMode [ /Server | /Plugin ]**

The **/MergeMode** option controls compatibility with pdfExpress versions prior to 1.0.5. In general, this option should always be included in any script file might be used with the pdfExpress server. It is ignored by the plug-in versions of pdfExpress.

The default value is **/Plugin**.

**/EnableBottomMediaBox [ true | false ]**

The **addPage** command (defined below) can be used to layer multiple PDF files. Normally, the layering only occurs *on top* of an **addPage** base page (with **\$Page.MediaBoxTop**). When this option is enabled, the layering can be specified underneath a base page, i.e., the base page is transparent and items can be located underneath it.

The **/EnableMediaBoxBottom** default is false since pdfExpress must preprocess each page under which a **MediaBoxBottom** could potentially occur. This

preprocessing can require considerable time and overhead, depending on the PDF file, and is therefore disabled unless required..

The default value is always false.

### **/AllPagesSupportMediaBox [ true | false ]**

The addPage command (defined below) can be used to layer multiple PDF files transparently, one over the top of the other. Normally, layering can occur only on PDF pages that have been marked up with pdfExpress prior to merging and that server as the base for the addPage.

When this option is true, the prior markup requirement is relaxed and any addPage base-page may be layered over. However, there is a computational expense associated with the preparation of any page onto which layering can occur. This preparation is typically accounted for on marked up pages but will cause perhaps significant additional computation when used with all pages.

The default value is false.

### **/AddPlatformSignature [ true | false ]**

When true, text is appended to any file name generated by pdfExpress. This option is used to allow pdfExpress to automatically identify the platform version of pdfExpress used to create the file.

The supported signatures added to the file names are `_WIN` for any windows plug-in version, `_MAC` for any Macintosh plug-in version, `_WSVR` for the windows and NT server version, and `_UNIX` for all unix server versions.

The default is false.

### **/ErrorReporting <<**

**/Log [ (filepath) /Stderr /Stdout /None ]**

**/Dialog [true | false]**

**/Abort [true | false] >>**

The defaults are `/Log /None`, `/Dialog true`, `/Abort true` for the plug-in versions and `/Log /Stderr`, `/Dialog false`, and `/Abort true` for the server version. Presently changing these options to values other than the defaults is not supported.

### **/ExpandPathMacros [ true | false ]**

When true, strings provided to pdfExpress containing path names are passed through a macro expander prior to being passed to the file system. This allows the

user to create defined names for directories and folders which can be changed universally in a script file simply by redefining the macro definition. The **/MacroDefs** entry (see below) defines a default set of macros. When this value is false, no path macros are expanded.

**/MacroDefs**    << ...pathDefs... >>

Each platform defines the following macros as defined below. Users can replace the **pathDefs** file with other values. User choices override the default values below.

PC (plug-in and server):

```
<< /_pathSep (\) /_thisDir (.\\) /_dotDot (..) /_parallel (..\\) >>
```

Unix:

```
<< /_pathSep (/) /_thisDir (..) /_dotDot (..) /_parallel (..) >>
```

Macintosh:

```
<< /_pathSep (:) /_thisDir () /_dotDot (:) /_parallel (:) >>
```

**Acrobat Syntax:**    [ /end ] merge

A merge engine initialized by the **/begin** command is terminated with the **/end** command. If the engine has not been initialized, invoking **/end** will generate an error.

**Acrobat Syntax:** [ /reset ] merge

The active merge engine is completely reset and terminated. All active carousels, pages and output streams are closed. The state of any the output streams is not guaranteed after a reset.

### 4.3 Invoking Other Script Files

**Acrobat Syntax:**    [ /execute (filename) ] merge

You can invoke another script file from within a pdfExpress script by use **/execute**. The file, represented by the platform specific string **filename**, is assumed to contain more pdfExpress commands (in either XML or Acrobat form). If filename contains a relative path name, the path is relative to the location that contained root .LIS or .LXS file opened by pdfExpress.

The invoked script file is processed within the context of the existing merge engine. This means that all existing carousels, output streams, etc. persist across the bounds of **/execute**.

## 4.4 Carousels

Input to the merge process for PDF files occurs via a *carousel*. A carousel is a logical name for a particular PDF file or dynamic PDF page at a particular time. The scope of the carousel extends lexically from a **/createCarousel** command to a matching **/deleteCarousel** command, i.e., once a carousel is created, it may be referenced by any intervening pdfExpress script, until the **/deleteCarousel** command is encountered. All carousels are closed by a **/reset** or **/end** command.

During execution of a merge a carousel name may be associated with a different file by first deleting and then creating a new carousel with the same name.

The merge pages are copied from currently active carousels to the output stream or streams. Only PDF pages loaded as part of carousels may be accessed during the merge by carousel references. During the merge process currently active carousels may be closed and opened or reopened.

Elements within PDF files associated with carousels are referenced with the *triple* notation, specifically *carouselName:pageNumber.namedElement*, or the abbreviated triple notation for referencing pages, specifically *carouselName:pageNumber*. This triple notation can be thought of as a path to a specific object on the PDF page. The *carouselName* associates a particular PDF file with the triple. Within that particular PDF file, the *pageNumber* resolves the triple to that specific page. Finally, the *namedElement* component resolves the triple to a specifically marked up object on that page.

The triple notation provides a powerful abstraction model that allows multiple PDF files, each marked up identically, to be used interchangeably. For example, a workflow might use high and low res versions of marked files simply by changing a path or macro reference.

Any number of carousels may be active at any given time within the merge engine subject to Acrobat, server, platform and/or memory limitations.

## 4.4.1 Creating Carousels

### Acrobat Syntax:

```
[ /createCarousel (carouselName) (pathToPDF)  
    << ...attributes... >>  
] merge  
or  
[ /createCarousel (carouselName) << pagespec >> ] merge
```

Prior to the first reference of a carousel name in a pdfExpress script the carousel must be created with the **/createCarousel** command. In the first form above, the carousel identified by *carouselName* is associated with the PDF file identified by *pathToPDF*. The attributes dictionary may be used to specify the following options:

### **/copyOnOpen [ true | false ]**

Normally a carousel is a copy of the Acrobat PDDoc associated with the PDF file. This copy is made in memory during the execution of a pdfExpress **/createCarousel**. A copy is made in order to eliminate any chance that operations within pdfExpress might corrupt the underlying PDDoc. If a file is referenced via by a single carousel within a script, **/copyOnOpen** need not be true. However, in cases where the same PDF file is accessed via multiple carousels or output streams simultaneously, **/copyOnOpen** should be true.

For small files, the overhead of copying the file is small. However, when file sizes are large (for example, with large, high resolution scans) or contain a large number of Cos objects, the overhead of making the copy can become expensive, particularly with the server version. To eliminate this overhead, this option is provided.

The default is true.

The second form of **/createCarousel** is used to create dynamic PDF pages. A dynamic PDF page is a page not associated with any file. Only users experienced with writing native PDF or PostScript should use this form. Certain printer features, such as selecting paper bin or gathering sheets into groups for stapling, can only be controlled via PostScript. Dynamic PDF pages offer a mechanism to control these aspects of the devices.

With this form, the **pagespec** dictionary must contain the following entry:

```
/Page << ...pagedef... >>
```

where the **pagedef** dictionary may contain the following entries:

## **/MediaBox [ bottom left top right ] - required**

The **/MediaBox** entry defines the size of the page to add. The bottom and left values should always be zero. Top is the height of the page in points, right the width in points. Note that pages are always created with a rotation of zero (0).

## **/Rotation *n***

The **/Rotation** entry is used to define the created pages' PDF rotation. Valid values for *n* are 0, 90, 180, and 270.

## **/Resources << ... resources dict... >>**

The **/Resources** entry can be used to create PDF resources to associate with the page. If the **/Resources** entry is not present, the following default **/Resources** dictionary is used (note that for Windows and server platforms the value of XXX after **/Encoding** is defined as WinAnsiEncoding and for Mac platforms the value of XXX after **/Encoding** is defined as MacRomanEncoding).

```
<<
  /ProcSet [ /PDF ]
  /ExtGState <<
    /GS1 <<
      /Type /ExtGState /SA false /SM 0.02 /TR /Identity
    >>
  >>
  /Font <<
    /H <<
      /Type /Font
      /Subtype /Type1
      /Encoding /XXX
      /BaseFont /Helvetica
    >>
    /HB <<
      /Type /Font
      /Subtype /Type1
      /Encoding /XXX
      /BaseFont /Helvetica-Bold
    >>
    /C <<
      /Type /Font
      /Subtype /Type1
      /Encoding /%s
      /BaseFont /Courier
    >>
    /CB <<
      /Type /Font
```

```

    /Subtype /Type1
    /Encoding /XXX
    /BaseFont /Courier-Bold
  >>
>>
>>

```

### **/Contents (string containing pdf commands)**

The **/Contents** entry can be used to specify PDF commands that will be used to mark the page. The default contents for a page are defined as:

```
0 0 0 1 K /GS1 gs 0 0 m S
```

These resources may be referenced by the pdf commands. For example, the /GS1 resource referenced by the gs command in the **/Contents** string is defined in the default resources.

```

/XObjects [
  [ /ObjectName << ... xobject dict ... >> (stream contents)
  ...
]

```

The **/XObjects** entry can be used to create dynamic XObjects that are added to any resources defined for a PDF page. The object list consists of one or more object list entries as defined above. Any XObject create in this fashion can be accessed via a **/Contents** entry via the PDF Do operation.

For example the following **/Page** is used to embed PostScript-based actions in a PDF page.

```

/Page
<<
  /MediaBox [ 0 0 792 612 ]
  /XObjects [
    [ /PagePick << /Type /XObject /Subtype /PS >>
(PDFVars/TermAll get exec end end
restore
featurebegin{
<< /MediaType \ (letterhead\ ) >> setpagedevice
}featurecleanup
PDFVars begin PDF begin PDFVars/InitAll get exec
save
/N14 <<
/SA false
>> /ExtGState defineRes pop
0 0 612 792 RC)

```

```
]
]
/Contents (/PagePick Do)
>>
```

These resources may be referenced by the pdf commands.

Carousel names should start with a letter and may only contain letters and numbers. This restriction is not enforced by the software.

## 4.4.2 Carousels and Security

When a document is protected with Acrobat security, you cannot extract pages from the corresponding Carousel nor can you access pages via the *(:file.pdf.n:)* notation within an **/addPage** command. pdfExpress (as well as the Acrobat menu item Document>Extract Pages) requires the ability to do this in order to move information from the carousel to the output stream. The only means to access pages from within a secure document is for pdfExpress to have the rights to save a copy of the document. With these rights, pdfExpress can save a non-secure copy of the document, reopen the non-secure copy, and use it in place of the secure PDF. The secure copy is deleted when the carousel is closed.

Using pdfExpress in this mode of operation raises several important security issues.

You must have “Modify Security Options” permission on any PDF file pdfExpress is to process. It is not enough to have the “Open Document” password.

pdfExpress must operate on a secure server such that the protected copy of the file is not generally accessible.

Any .LIS (or .LXS) file will contain the password for the document. You must make sure it is handled securely.

When pdfExpress is running, the unsecured version of the file will exist on the file system. Should pdfExpress terminate unexpectedly (say due to a bad script) the unsecured copy of the file may be left on the disk. Since pdfExpress creates temporary PDF files to hold the unsecured information on a scratch, you may not be able to locate unsecured copies.

PDF allows multiple types of security (presumably so other plug-ins can create and manage secure files). pdfExpress currently only supports “Standard” Acrobat/PDF security. “Standard” security is the security mode you get when choose “File>Save As” and choose the Security drop-down item.

Note that the password supplied in the `/createCarousel` command is for the “Open Document” security feature, *not the “Change Security Settings” options*. Since each document can have two passwords with Standard security, it is important to use the right one.

Security is manipulated in pdfExpress via the /createCarousel attributes dictionary. The following options control security.

### **/Password (password)**

The “Document Open” password is given as “password”. An error will be generated if the password is not correct and the PDF file has an “Document Open” password.

### **/SecurityType (Standard)**

The PDF file’s security is given as Standard. No other security type is supported. This is the same the default PDF file security. Both this and /Password are required to open a secured PDF.

## **4.4.3 Deleting a Carousel**

### **Acrobat Syntax: [ /deleteCarousel (*carouselName*) ] merge**

The named carousel is deleted. Subsequent references the carousel will result in errors. All resources associated with the carousel are freed.

## **4.5 Output Streams**

pdfExpress output is directed to a new PDF file through an output stream. A pdfExpress merge is the process of copying pages from one or more carousels, merging data into the copied pages, and copying the page to an output stream.

One or more output stream may be active at any given time. Each output stream is named. Output streams create one or more PDF files according to an output path which specifies the directory in which the create the file and a file name or a file template that defines how the file is to be named. A file template can be used instead of a single file name is used because pdfExpress jobs can often produce many tens or hundreds of thousands of pages of output and a single PDF file would be inappropriate for storing that many pages. File templates allow pdfExpress to specify an output directory in which pdfExpress creates files according to a well-defined template model. The user can specify the number of pages in each output file directly or indirectly.

As a destination for carousel pages there are no restrictions as on the number of pages nor their media sizes. This means that an output stream may contain mixed media sizes, e.g., 8.5” x 11.0” pages along with 11” x 17” pages. Note that such files may present problems when presented to an output device such as a printer.

## 4.5.1 Creating an Output Stream

### Acrobat Syntax:

[ /createOutputStream (*streamName*) <<*attributes*>> ] merge

An output stream named *streamName* is created. The **attributes** dictionary is defined to support the following entries:

#### **/SaveMode [ /FullGC | /Full | /Default ]**

The /SaveMode entry defines how a file that is created by an output stream will be saved. The /FullGC option removes all unreferenced content from the file prior to saving, and creates a new, complete PDF structure in the output file. This PDF will have all unused Cos references removed. This option is the most expensive in terms of time and computation but yields the smallest output file size. The /Full option saves the file, but does not remove duplicate or unused Cos objects. The resulting file size may be larger, but the save may be faster. The /Default option simply saves the file in the most convenient form for the merge engine. Note that this may only consist of appending a new xref index to the end of the file.

There is no guarantee that the file will not be removed and/or renamed during this process, so certain operating system functions such as file locking may interfere with the process.

The default is /FullGC.

#### **/OutputPath (path) – Required (formerly /OUTPUTPATH)**

Define a path name to be prepended to the output file or template name. An empty string (which still must be specified) means that the file is created where ever pdfExpress thinks the current working directory is. (Note that the notion of working directory differs from platform to platform, but generally is considered to be the last folder in which a PDF or script file was opened.)

In general, since backslash (\) is a special PDF string character, it must be escaped in all PDF strings. Thus the current folder (.\) must actually be entered in a pdfExpress script as (.\).

On the Macintosh, the colon is used to specify folder locations, e.g., Red:blue:green. A double colon (::) means the containing directory.

No default.

**/FileName (name) – mutually exclusive with /FileTemplate.**

If present, specifies the name of the output file (note that if the platform requires a file extension, it must be specified). Either /FileName or /FileTemplate must be specified.

No default.

**/FileTemplate (templatename) – (formerly /FILETEMPLATE).**

If present, specifies the format of a file template. A file template specifies how a file name will change from output segment to output segment (note that output segments are current not supported). In general, a file template is a file name which must contain a template specifier and no file extension.

A template specifier is a sequence of nines and letter A's (9) and (A) enclosed between two percent signs, e.g., %99%. For each new output segment, the resulting file name is incremented (a nine specification increments from 0 to 9, a letter A specification from A to Z). Thus for a specifier of the form %99%, the first segment is 00, the second 01, and so on.

Pages added to an output stream via a single the **/addPage** command using the multi-page add syntax will be kept together: breaks between segments will occur only between integral **/addPage** commands.

No default.

Output stream names must start with a letter and may only contain letters and numbers. This restriction is not enforced by the software.

## 4.5.2 Appending to an Existing Output Stream

**Acrobat Syntax: [ /appendOutputStream (*streamName*) <<*attributes*>> ] merge**

The named stream is created by opening an existing PDF file and appending pages to it. Otherwise, the functionality is identical to /createOutputStream except for the following:

**/FileTemplate (templatename)**

This option cannot be used with /appendOutputStream.

### 4.5.3 Terminating an Output Stream

**Acrobat Syntax:** [ `/closeAndDeleteOutputStream (streamName)` ] merge

The named stream is flushed and closed. All output pages added via `/addPage` are appended before the stream is closed. Subsequent attempts to add pages to the stream will result in errors.

Note that depend on platform and program type (server versus plug-in) the exact limits of how many pages may be added to a file will vary. In general, no output will be appended to the output file until this command is encountered in the script file. However, a large amount of cached PDF may exist in the file system in a temporary folder or in virtual memory.

The user must ensure that there is enough storage available to store the entire job on disk during its execution.

## 4.6 Output Streams and Security

Security options for output streams are supplied as part of the attributes dictionary present in the `/createOutputStream` or `/appendOutputStream` commands. Presently, output streams accessed by `/appendOutputStream` and which are secured cannot be modified by pdfExpress.

The following modifiers may be supplied to the output stream attributes dictionary:

#### **`/OpenPassword (xxx)`**

Sets the “Document Open” security password for all output files (including those generated by templates) to “xxx”. If not present, then the created PDF file will not have “Document Open” security.

#### **`/ModifySecurityPassword (xxx)`**

Sets the “Modify Security” password for all output files (including those generated by templates) to “xxx”. If not present, then the created PDF file will not have “Modify Security” security, i.e., the user will not be able to change the security on the document.

#### **`/SecurityType (Standard)`**

The PDF file’s security is made Standard. No other security type is supported. This is the same the default PDF file security. Both this and `/OpenPassword` or `/SecurityType` are required to create a secured PDF.

## **/Permissions [ ...permission list... ]**

This modifies the actual specific PDF permissions granted by supply the above passwords. For example, to supply the Open and Secure permissions, specify /Permissions [ /Open /Secure ]. Permissible permission values for the permission list are listed below. The absence of the permission value explicitly denies the user of the create output PDF that permission.

- /Open**            Should be present in the permission list if the user will be able to open and decrypt the PDF file created by the associated output stream.
- /Secure**        Present in the permission list if the user will be allowed to change the security settings associated with the PDF created by the associated output stream.
- /Print**          Present in the permission list if the user will be allowed to print PDF document created by the associated output stream (this includes File>Print and Page Setup).
- /Edit**            Present in the permissions list if the user will be allowed to modify the content (other than add or change text notes) in the PDF file created by the associated output stream.
- /Copy**           Present in the permission list if the user will be allowed to copy text or graphics out of a page in the PDF file created by the associated output stream.
- /EditNotes**    Present in the permission list if the user will be allowed add, change, or delete text notes in the PDF file created by the associated output stream.
- /SaveAs**        Present in the permission list if the user will be allowed to perform SaveAs (if both Edit and EditNotes are disallowed, Save will be disabled, but SaveAs is still allowed) in the PDF file created by the associated output stream.

## **4.7 Adding Pages and Replacing Values**

The process of adding pages to an output stream is accomplished via the *addPage* command. The command is used to *copy* one or more pages from one or more carousels, dynamic file references, or dynamic page definitions, to *perform* image and textual *replacements*, and to *append* the resulting pages to the output stream. This command is the only command that is allowed to modify an output stream.

The **/addPage** command consists of three parts: the destination output stream, a list of one or more source PDF pages, and a dictionary of named replacement data that will be

used to modify any marked up or layered elements in or on the source pages to be copied.

Items may be replace singly or with blocks, depending on the original mark up.

All named replacement data is fully specified by text, regardless of whether text or graphical elements are being replaced.

For textual replacement the exact font, font size, inter-character width, and inter-word width of the text being replaced is preserved between the mark up and the replacement. (Please see the document *Preparing Files for pdfExpress* for a detailed description of text replacement issues.) In general, replacement text will look exactly like the original text. For textual replacements involving blocks, the space between the blocks is translated into the leading of the replacement text.

Note that since pdfExpress is not based on Acrobat forms, there are not font or other limitations on what can be replaced.

For purposes of replacement, pdfExpress defines a *graphic* as a bit-mapped black and white or color image. This is to distinguish between all bit-mapped images and PDF-based image operators for generating lines, rectangles, and so on. In pdfExpress only graphics may be replaced by other graphics or PDF pages during the merge process.

Graphic replacement occurs by providing pdfExpress a textual *reference* to a replacement graphic or PDF page. Graphic references are textual in nature and specify either a marked element in an existing, active carousel, a page in an existing PDF file, or a dynamic PDF page.

Unlike text, the default behavior of graphic to graphic replacement of images is *scale to fit*. This means that replacement graphics are automatically scaled to fit the original graphic. For replacement of graphics by PDF pages, the default behavior is to replace the graphic with the PDF page *at actual size*. In both cases, a CTM may be used to override any default scaling.

PDF graphics are by default rectangular in nature with an explicit bounding box size. If a replacement graphic is does not have the exact aspect ratio of the original graphic, the *scale to fit* behavior will likely produce undesirable effects.

Both text and graphic replacement may be single-item-based or block-based. Single item replacement involves the one-to-one replacement of a marked original element. Block-based replacement involves replacing a single initial element as well as placing subsequent additional elements based on the example offset provided in the original markup.

In pdfExpress, all single-item replacements are described by a String or a Dictionary. For example, suppose that a date on a letter has been marked up and named "date". To replace that date with "June 1, 2000" the string (June 1, 2000) would be supplied to the addPage command. Similarly, to replace a graphic marked "signature" with the signature marked "President" on page five (5) of carousel "Signatures", the string (Signatures:5:President) would be used.

If, however, the markup involved a “block” replacement, i.e., a replacement where an initial element and an iteration element were linked to define a iteration or replication block, then multiple replacement values would need to be specified. In pdfExpress, anywhere a block replacement occurs, an Array of strings (or dictionaries) would be used.

For example, suppose an address block in a letter has been created called “block.address”. To supply multiple replacement values, i.e., to have a dynamically growing address field, individual values for each line of the address are supplied as strings that are in turn part of an array. The array then represents the complete replacement “value”. So for example, the array [ (Anytown, USA 99999) (1234 Main Street) (A. B. Blank) ] represents an address block which might look like:

```
A. B. Blank  
1234 Main Street  
Anytown, USA 99999
```

in a merged letter.

## 4.7.1 Naming and Referencing Characters

Most standard Adobe Type 1 fonts (as well as other font types such as True Type) support a naming convention for characters. This naming convention is documented in the PostScript and PDF reference manuals supplied by Adobe.

Each character glyph (the image of the character) has a unique name. For ASCII characters represented in the United States, these name correspond roughly to the characters on the keyboard. Thus the character N is named N, z is named Z and so on. Other, special characters such as © have names as well. For example, © is named, surprisingly enough, copyright. In pdfExpress, these characters are represented by /glyphName, e.g., N is /N, copyright is /copyright and so forth. Such names are always case sensitive.

The strings used in the previous example to represent the canonical ASCII use of values, e.g. Mr. A. B. Blank is represented by the string (Mr. A. B. Blank). However, there are cases where the canonical ASCII value of a character, e.g., ©, is not know, i.e., you don't know what value to put between the parends. Such characters may be referenced by name or numeric position in the font encoding within pdfExpress. Thus the copyright symbol © is named /copyright. To create a string using the symbol name of the character, an array is used. Thus string AB©D is represented as [ /A /B /copyright /D ]. Arrays of glyph names can also contain string. Thus AB©D could also be represented as [ (AB) /copyright (D) ].

Characters can also be represented by an octal (base 8) value. Thus the letter A can be represented in pdfExpress as (A), (\101) and [ /A ]. Users of pdfExpress should adopt the following rules of thumb for representing characters.

### 4.7.1.1 Rules for Referencing Characters

In the US, the ASCII character set may be used the a pdfExpress string to access the traditional ASCII printing characters. Thus (ABC) represents ABC. Strings containing non-printing or special characters such as copyright should be created with the array form of a string, e.g., AB©D is represented as [ (AB) /copyright (D) ] as defined above.

Symbolic character references such as /copyright *tend* to be portable across fonts, i.e., most Type 1 fonts, for example, adopt the Adobe glyph naming conventions described in the PDF and PostScript manuals. A less portable choice is to reference the character by an octal value. This is less portable and tends to vary through each distillation or by each application creating the original PostScript or PDF.

Note that characters outside the usual ASCII (PC keyboard) range may not be fully embedded in a PDF file. The only way to ensure that the character is there is to actually embed it somewhere in the PDF file. (If you don't want to see it, you can make the string that it is part of invisible with the markup process.)

Characters that are not fully embedded have a width of zero (0). This means that though the character may print, the next character will over print the previous one.

If you are outside the US but still use an ASCII-based roman character set, e.g., Europe, it is recommended to use either all glyph names or a mix of strings and glyph names. The choice will depend on the font used, whether the font or fonts have reliably named glyphs, and the application creating the file. Experimentation may be required to create a reliable replacement model.

Users in Eastern Europe that use non-ASCII characters sets, e.g., cyrillic, should always use arrays of glyph names. Care must be taken to ensure the entire character set is embedded in the file and also that the creating application embeds the font such that the expected character names, .e.g, wjedi, are used consistently.

Currently we have no known examples of pdfExpress with Asain and vertical fonts.

### 4.7.2 Adding Information to Replacement Values

Anywhere a string value can be used in a replacement a dictionary may also be used. Dictionaries used in this context must have at least a **/Value** entry where the value portion of the name/value pair is a string value as described above. Thus (A. B. Blank) and << /Value (A. B. Blank) >> are exactly equivalent,. In addition to the /Value entry, the dictionary may also contain a **/CTM** entry. The value of a /CTM entry is an array of six (6) values: /CTM [ 1 0 0 1 0 0 ]. The CTM entry is used to specify PostScript- and PDF-style *Current Transformation Matrix* information to the replacement operation.

Although the operation of CTM replacements is complex CTM provides a way to perform exact location placement, scaling, rotation, and skewing of text and graphic elements. In a replacement dictionary a specified CTM is concatenated to the current, internal PDF CTM prior to placing text or graphics.

Additional options for controlling replacements are also available via **/Value**. The options are defined below as part of **/addPage**.

Graphic replacement strings are used to specify either graphic elements from existing PDF's in active carousels, pages from specific files located on the operating systems file structure, or pages created dynamically. References to existing carousels follow the triple notation described earlier: (*carouselName:PageNo:name*). *CarouselName* is the case sensitive name of a previously declared carousel, *PageNo* is the one-based (one (1) is the first page, two (2) is the second page, and so on) page within the file, and *name* is the name of a marked graphic on that page in the named carousel. Triple references such as these are independent of the actual PDF file loaded into a given carousel at any point in time. Using this notation the user can access all available images within the pdfExpress environment.

References to external files are defined by strings with the following format: (*:pathToFile:pageNo:*). In these references *pathToFile* is an operating system specific reference to a PDF, JPEG, BMP, WMF or uncompressed TIFF file. Files on the Windows/NT version of pdfExpress (server or plug-in) are typed by their extension: .PDF implies a PDF file, .JPG implies a JPEG file, .TIF implies a TIFF file, and .BMP implies a Microsoft .BMP file, and .WMF implies Windows Meta File. Bitmaps loaded with this form are converted to RGB. *PageNo* must be one (1) for all file types except PDF (access to multiple page TIFF files is not supported). For PDF files the *pageNo* specifies the page within the PDF file to use. The reference must include the complete, properly cased file name. If relative file names are used, e.g., *../foo/smith.pdf*, please note that the file name is relative to the current directory with respect to the script file.

Dynamic page references are similar to dynamic carousel pages described previously, but can occur where ever a **/Value** construct can occur in an **/addPage** command.

### 4.7.3 Adding a Page

#### Acrobat Syntax:

```
[ /addPage (streamName) [pagerefs] << substdict >> ] merge
```

The **/addPage** command copies the page or pages referenced in **pagerefs** to the output stream specified by the **streamName**. As the page is copied, each marked element in each page is looked up by name (case sensitive) in the supplied *substdict*. If a string, array or dictionary value is found for that name, then the value of that string or array (or value of a **/Value** entry in the case of a dictionary) is applied as a replacement after the pages is copied.

For any item on the pages to be copied that does not exist in **substdict**, the resulting copy of the page will contain the original, marked up element if and only if the item was **not** marked invisible. If the item was marked invisible, then nothing will appear in place of that object in the output.

Multiple item replacements must be supply an value array in the **subdict**. Single item replacements must be supplied with a non-array value.

The lookup process is unordered and no reliance on the fact that a particular element is looked up before or after some other element should be made.

The **pagerefs** section of the **/addPage** command supports the following types of page references:

### **Individual Carousel Page – (carName:page)**

With this format, a specific page (**page**) is copied from the carousel defined by **carName** (an error occurs if carName does not define a currently active carousel) to the output stream.

### **All Carousel Pages – (carName:\*)**

With this format, all pages in the PDF file represented by **carName** are copied, in order, to the output stream.

### **Specific Carousel Pages – (carName:1,3,5-8,11-\*)**

In this format, the specific pages indicated (1, 3, 5, 6, 7, 8, and all pages from 11 to the end of the document) are identified for appending. The pages identified must be present in the carousel. Pages can be supplied in any order.

Range values are inclusive and all pages must exist, i.e., you cannot say 1-12 for an eight page document. To represent the last page in a document use “\*”.

The **pagerefs** is an array of one or more page references as defined above. *There must be at least one page to copy from the pagerefs list to the output stream.*

Within the subdict, replacements are defined by the following general form:

### **/Name valueitem**

Where **Name** is the case-sensitive name of an item defined via mark up, one of the following special values: \$Page.MediaBoxTop or \$Page.MediaBoxBottom, or a reference (in Acrobat 4.0 and greater) to an Acrobat form field name. Form field names are of the form **Form.name** where *name* is the name of a field defined with the Acrobat 4.0 or greater forms system.

### 4.7.3.1 Name Replacements

When **Name** refers to a single or group-item replacement item defined by mark up, each time a page is copied from an entry in **pagerefs** (regardless of which **pagerefs** form is used) to the output stream a replacement of the object (text or graphics) occurs in the output stream according to **valueitem**. This means that all pages having an object of like type (graphics or text) defined will be replaced by the same replacement object. For example, if **/Picture (c1:1:myDog)** occurs in the **subdict**, then any page copied as part of an **/addPage** command in which **/Picture** occurs will have the corresponding picture element defined by mark up replaced with the image referenced by (c1:1:myDog).

### 4.7.3.2 MediaBox Replacements

In the pdfExpress Pro version of the plug-in (both platforms) as well as all server versions, `$Page.MediaBoxTop` and `$Page.MediaBoxBottom` define a multi-item replacement block object which is the entire page. Values supplied to either of these named elements will overlay the entire media area of the page. `MediaBoxTop` replacements occur over the top of (in front of) anything that exists in the pages media box (note that `MediaBoxTop` replacements only make sense if the replacing item or items contain clear areas which allow the underlying page to appear). Similarly, `MediaBoxBottom` replacements occur underneath the base page and only make sense if the base page has areas where the underlying replacement shows through.

For media box replacements, the following example demonstrates how output layering occurs. Suppose an **/addPage** command is defined as follows:

```
[ /addPage (oStream) [(c1:1)] << ...
  /$Page.MediaBoxBottom [
    (:fileA.pdf:1:)
    (:fileA.pdf:2:)
    (:fileA.pdf:3:) ]
  /$Page.MediaBoxTop [
    (:fileB.pdf:1:)
    (:fileB.pdf:2:)
    (:fileB.pdf:2:) ] ... >>
```

The resulting page added to `oStream` will look as follows: The pages 1, 2 and 3 from `fileA.pdf` will be overlaid exactly on top of each other, in order. Next, the page represented by (c1:1) will be overlaid, finally the pages 1, 2 and 3 from `fileB.pdf` will be overlaid, in order on the result of the previous overlays.

The resulting PDF will effectively represent a lamination of the three sets of pages, exactly as if each page were printed on a transparency and a stack made of the transparencies were converted to a PDF page. (Note that while areas would appear as the color white on the transparencies, whether as part of an image, text or line drawing. Since most

output devices do not create white output, but rather leave no output so that the underlying white paper substrate shows through, the behavior cannot be easily duplicated in reality with a printer and transparent paper.)

MediaBox replacements can also be used with `/CTM` (defined in a subsequent section) to position PDF files as part of the layering process. This process allows the placed PDF files to act as layout components.

```
[ /addPage (oStream) [(c1:1)] << ...
  /$Page.MediaBoxBottom [
    << /Value (:fileA.pdf:1:) /CTM [ 1 0 0 1 144 144 ] >>
    << /Value (:fileA.pdf:2:) /CTM [ .5 0 0 .5 72 72 ] >>
  ] ... >>
```

### 4.7.3.3 Form Element Replacements

As shipped from Adobe Acrobat supports forms. Typically Acrobat forms are populated with FDF-based tools in web browsers. PdfExpress provides a mechanism to populate form elements directly from within a pdfExpress script. A subdict containing named elements prefixed with form dot (Form.) are presumed to reference Acrobat form entities with that name, i.e., **/Form.fieldOne (Hello)** replaces the Acrobat form field **fieldOne** with the string value Hello.

Replacement values in this format can only replace string-based form elements, replacement of Acrobat graphic elements, and other, non-text items, is not supported.

Form element replacement allows pdfExpress to create active Acrobat forms which can be personalized both in appearance through traditional use of pdfExpress as well as their interactivensness through Acrobat forms.

### 4.7.3.4 Special Replacements

The **/addPage** also supports several special elements within in the **subdict**. Replacement items which are specified by **/Value** as defined above support the following modifiers which are included in the dictionary along with **/Value**.

#### **/CTM [ a b c d h v ]**

The **/CTM** modifier supplies a PostScript-style CTM transformation to the object being placed. The value of the CTM is calculated relative to the object begin replaced and is applied to the replacing object, not the object doing the replacement.

The object being replaced is considered to be of unit (1) height and width and the replacing object is considered to be whatever size it actual is.

For example, replacing a 1 x 1 inch picture with another 1 x 1 inch picture and specifying a specific transformation with CTM to apply to the replacement so that the image is simply replaced, i.e., no scale, skew, rotation, etc. is to apply, would be defined as: `/CTM [ .0138 0 0 .0138 0 0 ]`. The reason .0138 is used rather than 1 is that the object to be replaced is considered to be a unitless 1 x 1 size and the replacing picture is considered to be 72 points high and wide. The value of 1/72 is 0.0138. If the `/CTM [ 1 0 0 1 0 0 ]` were used instead, then the replacing object would be scaled to 72 times its original height and width.

All values are specified relative to the replaced objects size, thus an **h** value of 1 means to move 1 times the replaced objects size in the appropriate **h** direction and an **a** value of 2 means to scale the replacing object to twice its size in the appropriate dimension.

No default.

### **`/xStep [ fixed | (string) ]`**

This construct allows the user to override the x step distance for iterated replacements. A fixed value is assumed to be in points. If a string is supplied, the string must be of the form (number units) where number is expressed in units. Only inches (as in) and points (as pt) are supported. Thus (5 in) means step in the x direction five (5) inches.

Default value is the y distance specified with the iteration block's definition at mark up time.

### **`/yStep [ fixed | (string) ]`**

Similar to xStep, but for movement in the y direction.

Default value is the y distance specified with the iteration block's definition at mark up time.

### **`/xStepWidthPos (string)`**

### **`/xStepWidthNeg (string)`**

Applies only to items in a text iteration block. Similar to `/xStep`, but moves the width (positive for `..Pos` and negative for `..Neg`) of the string as defined by the font, font height, character spacing, etc. of the current iteration.

No default.

### **/DeltaMask [ bottom left top right ]**

Applies only to placed PDF files. When a replacement occurs with this modifier, no rendering occurs between the actual size of the object (its media box) and the distance specified by a new media box where bottom and left are added to the original media box and top and right are subtracted from the original media box.

No default.

### **/ ForcePageRotation *n***

If present, the PDF rotation of the page, regardless of the setting in the file being placed, is forced to *n*. The value *n* may be 0, 90, 180, or 270.

No default.

### **/MatchOutputPage *n***

If present, the value *n* is taken to be a page number in the output stream associated with the /addPage command. The replacement element containing /MatchOutputPage will be ignored unless the number of the page being added to the output stream by /addPage is *n*. If the output stream already has *n* pages, the element will never be added.

By default all /addPage replacement elements are added unless this option is present.

### **/MatchInputPage *n***

When this option is present in an /addPage substitution dictionary, pages copied from the carousel or other source are considered to have an order (1..n).

For example, if the command ... /addPage (out) [(carousel:1,4,5)] << ... / MatchInputPage 2 ... were encountered, the /addPage command appends three pages from *carousel* to the output stream. Thus page one (1) from *carousel* is appended first, page four (4) from the *carousel* is appended second, and page five (5) from *carousel* is appended third.

For a given replacement element, when *n* matches the *n*th page in the order, the replacement element is evaluated. If the output stream already has *n* pages, the element will never be added.

By default all /addPage replacement elements are added unless this option is present.

## 4.8 Performance Considerations

In order to obtain acceptable performance from pdfExpress, it is necessary to consider how PDF files are represented.

In general, performance is tied to memory usage. The more memory available, the faster the merge process. However, there are certain known operations that are not efficient in all situations:

1. Inserting a PDF page with a MediaBox or **/add Page** from a carousel with a large number of pages.
2. Creating an output stream with more than several hundred to thousand pages.
3. Merging pages which contain copies of fonts is generally less efficient than merging those without.

The underlying PDF Library used by pdfExpress is different in the server than the one used by the plug-in.

Both PDF Libraries exhibit different performance characteristics. The pdfExpress server (stand alone executable) is generally faster for operations involving a limited number of input and output pages. The server is also more sensitive to adding pages from carousels and files containing a large number of pages. The server also has a sharper performance knee, i.e., once the server consumes the available resources its performance deteriorates rapidly.

The plug-in base versions tends to have smoother performance characteristics in general, though it has a sharp performance knee as well.

In general, one needs to experiment with different techniques to accomplish a particular result: both in terms of performance and technique.

## 4.9 Understanding CTM's

The CTM or Current Transformation Matrix defines a way for the user to translate one coordinate system into another. CTM's are represented by a matrix.

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Transformations through the matrix are defined by the following formulas:

$$x' = ax + cy + t_x$$

$$y' = bx + dy + t_y$$

CTM's start out assuming a coordinate system based with (0,0) in the lower left hand corner of the page. Successive transformation may change or move that origin.

The values  $a$ ,  $b$ ,  $c$ ,  $d$ ,  $t_x$  and  $t_y$  are represented in PostScript, PDF and pdfExpress by a list of the six values. In pdfExpress the values are stored as an array of six numbers.

$$[ a b c d t_x t_y ]$$

In practice, the six values can be thought of as performing any or all of three graphic transformations to a give image or PDF page. These transformations are scaling, translation and rotation.

Scaling means making the unit distance in the coordnate system larger or smaller. Hence scaling by .5 makes everything 1/2 the size it normally is. Similarly, scaling by 3 makes everything three times its normal size.

Translation is the process of moving the origin of the coordinate system. Translating the coordinate system by 72, 72 means the origin is moved up and to the left 72 points.

Rotation is the process of rotating the coordinate system about the origin.

#### **4.9.1 Default Image Processing**

TBD

#### **4.9.2 Using CTM's**

TBD