

Part I

Basics



1

Overview of Adobe Graphics Server

This chapter provides a very general overview of the architecture of Adobe Graphics Server (AGS), how it works, what file formats it supports, and the underlying standards, technologies, and protocols that it uses.

The chapter contains the following sections:

Product Summary	starting on page 21
Format Summary	starting on page 23
Web-Related Standards and Formats	starting on page 26

Product Summary

AGS provides you with functionality you can use to programmatically request a particular set of operations on images and documents. This functionality falls into two main areas: specifying what manipulations AGS should perform, and invoking AGS to perform those manipulations.

AGS Commands

AGS provides an XML command language that you use to operate on input data you provide. These commands provide a wide range of manipulations, allowing you to:

- Modify, add to, and combine input data in various ways.
- Replace variables within data of certain types.
- Add data from XML sources and extract information in XML format.
- Modify metadata contained in the input.
- Convert between file formats; for example, from one image format to another, or between image and document formats.

You create command sets (in files or other language constructs such as strings) that contain a series of these commands. These commands instruct AGS to load input data and perform specific operations on it. After the desired transformations are done, you can save the results in one of several ways.

You can transform a single input file in a number of ways, creating multiple result files from a single input. You can also provide the same input file to the same set of commands, but use different associated data, such as variable replacement data, each time, resulting in a number of different result files from a single input file. In the latter case, input files are also sometimes called *templates*.

The information you need to construct command sets, and a list of file formats you can load, convert, and save are described in [Chapter 3, “AGS Command Basics.”](#) The AGS commands are described in detail in the *Adobe Graphics Server Command Reference*.

AGS Requests

You invoke AGS by sending it a request. A request consists of a set of commands, one or more input files, and some other optional information. AGS provides three APIs you can use to send requests: a Java, Perl, and COM API. You can also construct your own HTTP requests and send them directly to the AGS Web service. AGS also provides a convenient command line interface for executing commands during testing and prototyping.

You can specify in a request how you want AGS to return the results of its processing to you, either in a response, or by writing result files to the file system on which AGS is running.

For more information on how to send requests to AGS and process the results, see [Chapter 2, “Invoking AGS.”](#) The command line interface, the Web service interface, and Java, Perl, and COM APIs are described in detail in the *Adobe Graphics Server API Reference*.

Manipulating Graphics Input

AGS loads graphics files from a number of different formats, as listed in [Table 1.1 on page 24](#).

SVG files are loaded as SVG content. You can replace text and graph data in SVG content, as well as other operations. You can also convert SVG content to Raster content.

All other image formats are converted internally to the PSD-like format called *Raster content*. You can manipulate Raster content in a variety of ways. You can:

- Change the size of an image, crop or trim it, or change its orientation.
- Add, remove, position, or flatten layers.
- Change the color mapping in the image.
- Replace the pixels in a pixel layer and the text in a text layer.

You can define graphic template files that contain variables. You use ImageReady to create variables in PSD files, and Illustrator to create variables in SVG files. You can bind variables to images, text, SVG graphs and individual SVG objects, and also use them to toggle the visibility of objects or layers. You can then use AGS to replace the variable values in those templates. AGS provides a very flexible mechanism for automatic variable replacement in various circumstances.

You can convert document content (PDF and PostScript®) to image content (Raster and SVG), and image content to document content (PDF and PostScript). You can save Raster content to a variety of formats, as listed in [Table 1.2 on page 24](#).

For More Information

- For details on the general operations available on Raster content, see [Chapter 5, “Basic Image Manipulations.”](#)
- For details on the general operations available on SVG content, see [Chapter 8, “Creating Dynamic Graphs in SVG Files”](#) and [Chapter 9, “Operations Specific to SVG Files.”](#)
- For details on replacing text in SVG and Raster content, see [Chapter 7, “Replacing Variables in SVG and PSD Files.”](#)
- For details on variable replacement, see [Chapter 7, “Replacing Variables in SVG and PSD Files.”](#)
- For details on converting between formats, see [Chapter 10, “Converting and Saving PDF Documents”](#) and [“Treating PostScript and PDF Content as Images”](#) on page 104.

Manipulating Metadata

AGS uses Extensible Metadata Platform technology (XMP) to handle metadata associated with most of the file formats it accepts. AGS can read and write metadata values in both XMP and in the EXIF and IPTC formats defined for PSD, TIFF, and JPEG. It also handles metadata in PDF files.

You can export metadata from a file to examine or copy it, add, delete, or modify metadata properties for most file formats, and save files with metadata. AGS generally preserves existing metadata for those formats that do not support metadata manipulation.

AGS adds certain metadata properties to some files that it handles, to record data such as the date the file was modified and the fact that the file was handled by AGS.

For more information, see [Chapter 4, “Working with File Metadata.”](#)

Format Summary

When you load a file into AGS, it becomes AGS *content*. Content types do not correspond exactly to file formats; image data from most image formats is converted on load to the PSD-like *Raster content type*, and some content types can store more than one data format. For a complete explanation of the concept of AGS content and the relationship of content types to file formats, see [Chapter 3, “AGS Command Basics.”](#)

Input, Output, and Metadata Formats

The following table summarizes the files formats that AGS can read and shows the content type in which each format is loaded.

TABLE 1.1 *Input file formats allowed by AGS*

File format	Content type	Typical usage
PSD TIFF GIF JPEG PNG	Raster	Raster image data to be manipulated with the graphics commands.
SVG	SVG	Vector image data to be manipulated with the graphics commands.
PDF	PDF	Document input to be transformed to an image format.
PostScript EPS	PostScript	Document input to be transformed to an image format.
XML	XML	Replacement data for SVG or PSD variables, or other miscellaneous uses.
XMP	XMP	Replacement metadata values to be added to appropriate types of content, or extracted metadata being retrieved.

The following table shows the content types AGS provides and the file formats which you can save from each content type.

TABLE 1.2 *Output file formats produced by AGS*

Content type	File format
Raster	PSD, TIFF, GIF, JPEG, PNG8/24, WBMP
SVG	SVG
PDF	PDF, Web-optimized PDF
PostScript	PostScript, encapsulated PostScript (EPS), Photoshop EPS
XML	XML
XMP	XMP

The following table shows the metadata formats that AGS preserves and supports, and shows which file formats and content types can contain which metadata formats:

TABLE 1.3 Metadata formats supported and preserved by AGS

Metadata formats supported	File formats	Content type
EXIF, IPTC	PSD, TIFF, JPEG	Raster
XMP	PSD, TIFF, JPEG, GIF, PNG	Raster
	SVG	SVG
	PDF	PDF
	Photoshop EPS	PostScript

For files containing metadata in EXIF or IPTC formats, those values are preserved in XMP format when the file is loaded. When you save Raster content to a PSD or TIFF file, AGS writes the metadata in all three formats. When you save Raster content to a JPEG file, AGS can write metadata in none, any, or all of the formats.

For Raster, SVG, and PDF content, you can directly manipulate metadata only in XMP format. You can change, delete, or add properties. You can also export or import XMP metadata to or from an XMP file.

For Photoshop EPS, existing XMP metadata is preserved but you cannot manipulate it. Metadata is not preserved for other kinds of PostScript content.

Format Conversions

Some file formats are transformed on load (see [Table 1.1](#)) and some content types can be transformed on save (see [Table 1.2](#)). In addition, you can perform some explicit content type conversions on loaded content, as listed in [Table 1.4](#).

TABLE 1.4 Direct content type conversions supported by AGS

Raster	➡	EPS
	➡	PDF
SVG	➡	Raster
	➡	PDF
PDF	➡	Raster
PostScript EPS	➡	Raster

You can combine any number of these conversions. For example, you can load a JPEG file, which is loaded as Raster content, convert the Raster to PDF, and save out a PDF file. You can load an SVG file, convert it to Raster, convert it again to EPS, and save out an EPS file.

Web-Related Standards and Formats

AGS takes advantage of many Web-related standards. The command set is defined in XML, and you use XPath syntax to identify parts of Raster and SVG files. You use URIs (Uniform Resource Identifiers) to identify source files, and locations for result and log files. You can load XML data and use it to replace various kinds of data in the files you are processing. AGS also makes use of various Web-related formats such as SOAP, PDF, SVG, and XMP.

For an overview and a description of how these formats and protocols are supported and used in AGS, as well as information on formats and extensions that AGS defines or uses, see the *Adobe Graphics Server XML Grammars Reference*.

Many of these standards and formats are defined by the Internet Engineering Task Force (IETF) or the World Wide Web Consortium (W3C).

- For more information about SVG, SOAP, and other W3C standards, see the W3C site (www.w3.org).
- For more documentation of Adobe formats and protocols and Adobe extensions to standard formats and protocols, such as PDF, see the Adobe Web site (www.adobe.com).
- URIs are defined by IETF specifications.

2

Invoking AGS

This chapter contains a summary of the architecture of AGS and the various ways you can send it requests, and explains the underlying model of an AGS request and response. For more detailed information about these topics, see the *Adobe Graphics Server API Reference*. This chapter also provides a complete description of how you specify file locations using URIs in the API methods, on the command line, and in command attributes.

This chapter contains the following sections:

Deploying and Invoking AGS	starting on page 27
The Interface Model	starting on page 29
Details of the Objects in the Model	starting on page 31
How AGS Handles URIs	starting on page 35

Deploying and Invoking AGS

You can deploy AGS as a library and as a Web service. Once deployed, you have a choice of invocation mechanisms: the command line, one of three APIs (Java, Perl, or COM), or a direct connection to a Web service. [Table 2.1](#) summarizes how the deployment and invocation choices are related.

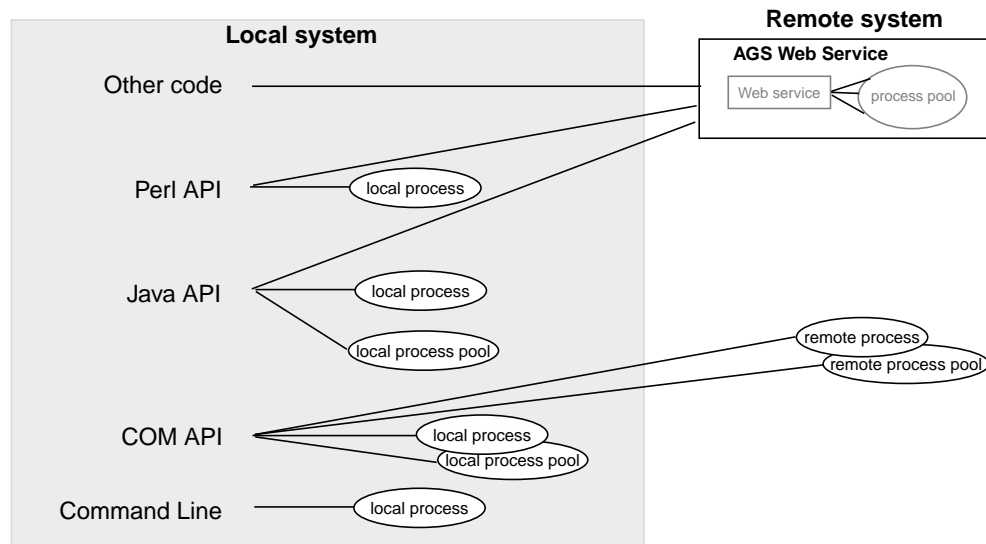
TABLE 2.1 *Deployment and invocation choices for AGS*

Deployment choice	Possible invocation choices				
	Java API	Perl API	COM API	Other	Command line
Library	✓	✓	✓		✓
Web service	✓	✓		✓	

When you install AGS as a library, you can access it locally from the command line or the Java, Perl, or COM APIs. Depending on how you configure your systems, it is possible to access AGS as a library remotely from COM.

When you install AGS as a Web service, you can access it from another system using the Java or Perl APIs, or directly by constructing your own HTTP request. Of course, local access to the Web service is also allowed.

[Figure 2.1](#) shows the same information as the table above. In addition, it shows where individual processes or process pools can be used.

FIGURE 2.1 Deployment and invocation choices for AGS

Which interface you choose to use to invoke AGS depends on the larger environment in which you are deploying AGS, and which language or mechanism you want to use to access AGS functionality. [Table 2.2](#) lists the four interfaces and the circumstances under which you might choose to use each one.

TABLE 2.2 Details of invocation choices for invoking AGS

Interface	Access modes	Usage scenarios
Command line	Local execution Windows & Solaris™	In an environment that supports command line mode, for simple invocation of AGS. You can work interactively directly on the command line, or from scripts that you can run from the command line or through some automatic mechanism such as cron.
Java API	Local and remote execution Windows and Solaris	In any situation where you can write Java code to invoke AGS, for example, in Java applications or servlets, or in JSP pages.
Perl API	Local and remote execution Windows and Solaris	In any situation where you can execute Perl scripts, for example, CGI scripts.
COM API	Local and remote execution Windows only	In any situation where you can execute code in any language with COM support, such as JavaScript, VBScript, Visual Basic, .NET, VBNET, Perl, Java, C, C++, and C#.
Web service	Remote execution	From any Web-service-enabled client.

When writing a set of commands, it is important to know how the commands will be submitted for processing. For example, the ways in which a command can refer to an input file or specify an output file location depend upon whether the request was made on the command line, an API, or directly to the Web service, and further on what data and information was included with the request.

There are also some interactions between specifications made on the command line or in a request, and those made within a set of commands. For example, you can specify a result location (where result files are written) on the command line, in the commands themselves, or both, or you can write results into a results object to be sent back to the client.

The Interface Model

While they differ in some respects, the invocation mechanisms are all based on a common model and rely on the same core AGS code that executes requests. This model defines objects that represent a server, request, response, and result. The three APIs implement this model directly, creating objects in their respective languages. The XML syntax for expressing AGS requests in an HTTP request is also based on the same object relationships. The command line interface more loosely matches the model.

The Objects in the Model

[Table 2.3](#) explains briefly what each of these conceptual objects represents, and the following sections provide a more detailed description of the objects.

TABLE 2.3 *The object model of the AGS interfaces*

Object	Represents
Server	An AGS process or service that executes a request and returns a response.
Request	A collection of all the necessary commands, input data, and request-specific settings needed to send a request to AGS.
Result	A unit of output from an executed command. The form of the output may be <i>result content</i> (current content when the command finishes), a <i>result file</i> (written to the local file system), or a named <i>result</i> that can be collected into a response object. You may get one or more output results from executing a request.
Response	A collection of log messages and results produced by an AGS server during the execution of a request, to be returned to the client. A response can contain zero or more results.

Interacting with AGS

In general, an interaction with AGS involves a number of steps. These steps vary depending on the interface you use to invoke AGS, as outlined in the following table. This table distinguishes between calling AGS from the command line, from one of the 3 APIs (Java, Perl, COM), or by sending an HTTP request directly to the AGS Web service (without using the Java or Perl API).

TABLE 2.4 Steps involved in interacting with AGS

Step	Interface	Details
<i>Creating an AGS Server object</i>	Java, Perl, COM APIs	The Java and Perl APIs provide constructors for server objects. Object construction is an integral part of COM.
	Web service direct	N/A
	Command Line	N/A
<i>Creating an AGS Request object</i>	Java, Perl, COM APIs	Java and Perl APIs provide request object constructors. Object construction is an integral part of COM. All three APIs provide methods for populating the request object with the required information.
	Web service direct	The HTTP message sent to the Web service is the request.
	Command Line	You can loosely view the information passed in the command line options as the request.
<i>Sending a request to a server</i>	Java, Perl, COM APIs	All 3 APIs provide an <code>execute</code> method in the server object's class that sends a request to a server.
	Web service direct	Sending an HTTP message to the Web service constitutes sending a request.
	Command Line	Invoking AGS on the command line "sends" the "request."
<i>Receiving a response from the server</i> (A response consists of log messages and result content.)	Java, Perl, COM APIs	The <code>execute</code> method returns a response object. Log messages are always included in the response object, but may also be written to a log file on the server's file system. Result content is returned either in the response object or written to the server's local file system.
	Web service direct	The Web service returns a response to the request message. The response contains log messages. Settings in the commands determine whether results are included in the response message or written to the Web service's file system.
	Command Line	Log messages are sent to the standard error stream. Result files are written to the local file system.

TABLE 2.4 Steps involved in interacting with AGS

Step	Interface	Details
<i>Processing the contents of a response</i>	Java, Perl, COM APIs	Methods are provided for extracting the log messages and results from the response object. If the results were written to the file system, you can inspect them with appropriate applications.
	Web service direct	You can use any method you choose to extract log message and result information from the HTTP response. If results were written to the file system, you can inspect them with appropriate applications.
	Command Line	You can redirect the standard error stream to a file for further examination of log messages. You can inspect result files with appropriate applications.
<i>Releasing a server when you are finished with it</i>	Java, Perl, COM APIs	With library deployment, Server objects persist until you explicitly release them, which releases their association with any AGS process and makes them available for garbage collection. (N/A for Web service deployment).
	Web service direct	N/A
	Command Line	N/A

Details of the Objects in the Model

The information in this section is a high level presentation of the details of the server, request, response, and result objects as given in the *Adobe Graphics Server API Reference*. An overview is provided here so that you can understand the context in which commands are executed.

Server Objects in the APIs

When you create a server object in any of the APIs, that object is associated, directly or indirectly, with an AGS process. Whether this association persists for the course of a single request or for the life of the object depends on what kind of Server object it is. [Table 2.5](#) shows the several kinds of server objects and how they are associated with AGS processes.

TABLE 2.5 *Kinds of server objects in the APIs*

Direct server object	Associated with a single AGS process on the same system on which the server object is created, for the lifetime of the object.
Pooled server object	Associated with a pool of AGS processes on the same system on which the server object is created. Each time you send the object a request, a different process from the pool can execute the request.
Web service server object	Associated with an AGS Web service, which uses a pool of AGS servers to execute requests it receives. The AGS Web service may exist on either the local system or a remote one. Each time you send the object a request, a different process from the pool can execute the request. You can establish a regular connection or a secure connection to the Web service when you create this type of object.
COM server objects	Associated with a single or pooled AGS process. Pooling is an inherent part of COM+, so you can create either single processes or process pools. Whether or not the process or pool is local or remote from the system on which the server object is created depends entirely on how your system or systems were configured during installation.

When you send a request to a server object that will execute remotely, there are some concerns you must consider, such as whether to include your input files in your request and whether to ask AGS to return your results in the response object or write them to the remote file system; these decisions can affect the speed at which the request is processed, due to increased transmission time. These concerns are discussed in detail in the *Adobe Graphics Server API Reference*.

Requests

When you invoke AGS, you must provide all the information it needs in order to do what you want it to do to your input files. This information includes specifying the input data on which you want AGS to operate, a set of operations to perform, and how to save the results of those operations. In addition, you can control certain aspects of how AGS behaves while processing this information. The collection of all necessary commands, input data, and settings needed for a single invocation of AGS is called a *request*.

The actual form the request takes depends on which interface you use to invoke AGS:

- If you invoke AGS from the command line, your request specified by the command line options and the contents of the command file you provide (if any—AGS can perform certain default operations on a file passed in on the command line when you do not specify a command file).
- If you invoke AGS from the Java, Perl, or COM API, you construct a request object provided by those APIs and put the appropriate information into it.

- If you invoke the Web service directly, you construct an appropriate HTTP request based on the WSDL (Web Services Description Language) for the Web service, using tools of your choice.

In general, the following pieces of information are part of an AGS request:

TABLE 2.6 *The kinds of information in an AGS request*

Information	Details
Input data	File content of any file type allowed by AGS.
AGS commands	A set of commands for AGS to execute on the input data.
Variable substitution data	Replacement values for variables defined in PSD and SVG files.
User-defined variables	Values for user-defined variables that can be referenced within commands. <i>Available only from the command line interface.</i>
Settings for AGS behavior	Values for certain parameters that control how AGS behaves while processing the request, including a base URI for the result location, whether result files should be overwritten, and the error behavior. These values can also be specified in the commands. If a value is specified both in the request and in the commands, the value set in the request takes precedence.

For details of how to construct requests for all of the invocation interfaces, see the *Adobe Graphics Server API Reference*.

Results and Responses

When you invoke AGS from one of the APIs or directly via the Web service, you can specify a result location in which AGS writes result files; this location must be locally accessible to the AGS process that executes the request. If you don't specify a result location, or otherwise save your results to the file system, result content is returned to you in a response. When you invoke AGS from the command line, all result files are written to the local file system.

Whenever you make a request from one of the APIs, AGS returns a response object. Response objects contain a (possibly empty) collection of log messages and a (possibly empty) collection of results. Log messages are simple strings, but results are represented by yet another object, the result.

A result object contains the file data for the result. It also contains information about that data, such as the name it was given in the commands when it was saved, an appropriate file extension, its MIME type, and if it is image data, the height and width of the image in pixels.

Logging

While an AGS process executes a request, it may output log information of various kinds. You can control the types of messages AGS logs and where it writes the log information.

Log Levels

Each log message starts with a prefix that corresponds to the level number. Levels (0-7) group messages according to their severity and importance in the system, as shown in [Table 2.7](#). You can set a parameter that causes AGS to report messages only at or below a particular level, using a command-line option or the appropriate API method when constructing a request.

TABLE 2.7 *Log levels and their meaning*

Level	Prefix	Use in AGS
0	[FATAL]	Not currently used.
1	[ALERT]	Not currently used.
2	[CRITICAL]	AGS has failed to complete the current request. No further commands will be executed for this request.
3	[ERROR]	An error occurred that makes it impossible for AGS to execute the current command. If the AGS process executing this request is configured to stop on errors, a log message at this level will be followed by a critical (level 2) message saying the request cannot be processed.
4	[WARNING]	A possibly incorrect but recoverable situation has occurred during the execution of a request. This is the default log level.
5	[NOTICE]	A significant event of interest occurred during execution.
6	[INFO]	A common but interesting event occurred during execution.
7	[DEBUG]	Not used.

Log Message Destinations

Where log messages are written varies across the invocation mechanisms:

- The command line sends all log messages to the standard error stream.
- The APIs allow you to specify a log destination for single server objects and for server pools. Regardless of whether or not you specify a log location, AGS always includes the log messages generated by your request in the response object. The APIs also provide a way to retrieve log messages for a request if AGS fails to finish executing it and does not return a response object.
- Log file location is configured by an administrator when the Web service is installed.

For details of the command-line options and API methods related to logging, see the *Adobe Graphics Server API Reference*.

How AGS Handles URIs

There are many places in the AGS interfaces and the AGS command language where you must supply a URI for the value of a parameter or attribute. This section explains the basic concepts you need to know in order to understand when absolute URIs are required, and how relative URIs are resolved. This explanation assumes the following definitions:

Absolute URI	A URI that begins with a scheme name, such as <code>file:///C:/myfiles/working</code>
Relative URI	<p>A URI that does not begin with a scheme name. Relative URIs can be absolute or relative paths, or simple names with no path information in them.</p> <p>The following are all relative URIs: <code>/tmp/logfiles</code>, <code>../myfiles/working</code>, <code>myfiles/working</code>, <code>image.psd</code></p>
Absolute path	<p>(Solaris) A path that begins with a forward slash, such as <code>/tmp</code></p> <p>(Windows) A path in one of three forms:</p> <ul style="list-style-type: none"> • A path that begins with a drive name, such as <code>C:/myfiles/working/image.psd</code> • A UNC path that starts with <code>//</code> • A UNC path that starts with <code>\\</code> <p>NOTE: AGS transforms <code>\\</code> to <code>//</code> on Windows.</p> <p>Absolute paths are relative URIs.</p>
Relative path	<p>(Solaris) A path that does not begin with a forward slash, or a simple name with no slashes or colons in it.</p> <p>(Windows) A path that does not begin with a drive name or with double slashes, as explained in the row above (Absolute path), or a simple name with no slashes or colons in it.</p> <p>Relative paths are relative URIs.</p>
Base input URI	An absolute URI used to resolve relative URIs for arguments that specify input data for AGS. The three APIs, the command line, and the core AGS process that executes requests all have their own base input URI.
Base output URI	An absolute URI used to resolve relative URIs for arguments that specify output data for AGS. The command line and the core AGS process that executes requests all have their own base output URI. (The SDK executables do not have base output URIs.)

AGS supports the following schemes for URIs:

TABLE 2.8 **Supported URI schemes**

<code>file:</code>	Supported in all APIs and the commands
<code>http:</code>	Supported only when specifying the URL of an AGS Web service. (Java and Perl APIs)
<code>https:</code>	Supported only when specifying the URL of an AGS Web service. (Java API only)
<code>adobe-content:</code>	Supported in some of the commands for specific purposes.

Base URIs for the APIs

Some of the methods in the Java, Perl, and COM APIs have parameters that take URIs. In some cases, you can specify relative URIs; in many cases you must provide absolute URIs.

Base Input URI: The SDKs establish their own base input URI, which is the current working directory in which the SDK was executed. Since you may or may not know what this location is, absolute URIs are usually recommended. The base input URI uses the `file:` scheme. You can always specify an absolute URI; in this case, the base input URI is ignored.

Base Output URI: The SDKs do not have a base output URI, so you must provide absolute URIs for all method parameters that identify output locations.

See the *Adobe Graphics Server API Reference* for method details. The entries for each of the method parameters in the APIs specify whether or not a relative URI is allowed.

Base URIs for the Command Line

Base Input and Output URIs: The command line establishes base input and output URIs, both of which are the current working directory, the directory in which you invoked the command line. Both base URIs use the `file:` scheme. Any relative URIs you specify as argument values on the command line are resolved against the appropriate base URI.

If you supply any absolute URIs as argument values on the command line, they must use the `file:` scheme. When you specify an absolute URI, the base URI is ignored.

All URIs you specify for the command line must identify files on the local file system.

See the *Adobe Graphics Server API Reference* for argument details. The entries for each of the command line arguments in that document specify whether or not a relative URI is allowed.

Base URIs for an AGS Process

Establishing the base URIs for an AGS process is more complicated than for the SDKs or the command line. The base input URI is influenced by which interface you use to submit the request and where the commands are for the request. The base output URI is influenced by which interface you use to submit the request and whether or not a result location is specified for the request. (You can specify a result location on the command line, in an API Request object, or in the commands themselves.)

The base URIs for an AGS process set the context for all relative URIs used in AGS commands.

API Invocation

Base Input URI: When you construct a Request object in any of the APIs, you can either include the commands themselves in the Request object, or you can specify a command file on the file system, which AGS reads to obtain the commands.

If you include the commands in the Request object, the base input URI is the Request object. All relative URIs for input attributes in commands, such as the `loadContent` command, must be either simple names with no path information in them (no slashes or colons) or absolute file: URIs of files on the system local to the AGS process.

If you specify a command file on the file system, the base input URI is the location of that command file, using the `file:` scheme. In this case, relative URIs for input attributes in commands, such as the `loadContent` command, can be simple file names or can contain path information (slashes or colons), and are resolved against the base input URI.

Base Output URI: When you construct a Request object in any of the APIs, you can specify a result location to which result files are written. The commands for the request can also specify a result location. (If it is specified in both places, the request setting takes precedence). The result location must be a directory locally visible to the AGS process that executes the request.

If you do not specify a result location, the base output URI is the Response object that is returned. All relative URIs for output attributes in commands, such as the `saveContent` and `saveOptimized` commands, must be either simple names with no path information in them (no slashes or colons) or absolute file: URIs of files that are written to the file system local to the AGS process.

If you specify a result location, the base output URI becomes the location specified. The result location must be specified using the `file:` scheme. All relative URIs given for output attributes in commands, such as the `saveContent` and `saveOptimized` commands, are resolved against this base output URI.

Regardless of what the base input or output URIs are, you can always specify an absolute URI, using the `file:` scheme, in which case the base URI is ignored.

Command Line Invocation

Base Input URI: When you submit a request to AGS from the command line, the base input URI is the location of the command file, if you specify one; otherwise, it is the current working directory. All relative URIs for input attributes in commands, such as the `loadContent` command, are resolved against this base input URI.

Base Output URI: When you submit a request to AGS from the command line, the base output URI is the result location if you specify one; otherwise it is the current working directory. All relative URIs for output attributes in commands, such as the `saveContent` and `saveOptimized` commands, are resolved against this base output URI.

Both base URIs use the `file:` scheme. Regardless of what the base input or output URIs are, you can always specify an absolute URI, using the `file:` scheme, in which case the base URI is ignored.

See the *Adobe Graphics Server Command Reference* for details on which commands have attributes that require URIs. The entries for each of these attributes specify whether or not a relative URI is allowed.

Base URI Summary

The following table summarizes the base URIs for the APIs (SDKs), the command line, and the core AGS processes (the context of the commands). All base URIs that are not the Request or Response objects use the `file:` scheme.

TABLE 2.9 *Base URIs for the AGS interfaces*

Interface where URIs occur		Condition	Base URI	
Java, Perl, COM APIs (Method calls)			Input:	Location where SDK was executed
			Output:	N/A
Command line (command line arguments)			Input:	Current working directory ¹
			Output:	Current working directory
AGS process (command attribute values)	Invoked from APIs	Commands in Request object	Input:	Request object
		Commands in file on AGS server's file system		Location of command file
		Result location not set	Output:	Response object
		Result location is set		Result location
	Invoked from command line	Command file supplied	Input:	Location of command file
		Command file not supplied		Current working directory
		Result location set	Output:	Result location
		Result location not set		Current working directory

1. The directory in which the command line is invoked.

3

AGS Command Basics

This chapter describes fundamental information you need to understand about how some of the basic AGS commands work and how to optimally combine commands in a command set. It explains how to specify and load input content, how AGS handles that content during request execution, and how to save output content. It describes the file formats, content types, conversions, and transformations that AGS supports. It also describes the techniques and mechanisms for specifying and working with content in commands, including how to handle metadata in the input data, and the ways in which you can specify attribute values for the AGS commands.

This chapter contains the following sections:

Overview	starting on page 39
Content Holders and Types	starting on page 40
Loading Content	starting on page 45
Converting Content	starting on page 48
Saving Content	starting on page 50
Referenced Attributes Values	starting on page 56
General Commands	starting on page 57

Overview

[Chapter 2](#) explained the various parts of an AGS request, one of which was a set of commands that direct AGS to perform a series of operations on specified input. You can specify this command sequence in various forms (stream, string, filename, or URI) depending on which interface you use to invoke AGS; the general term *command set* applies to all of these forms.

The AGS commands are defined using an XML syntax. A command is an element, and its parameters are its attributes. A command set contains a sequence of AGS commands in a command element. Observe the following conventions when writing command sets:

- The command set is an XML construct, and must conform to WC3 standards for well-formed XML documents.

- Indicate the text encoding in the first line of the command set:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

AGS supports UTF-8 and Shift-JIS text encoding. AGS also reads UTF-16 files, but converts them to UTF-8, and writes them to UTF-8.

- AGS commands are case sensitive.

- Comments can be included in a command set. The format for comments is:

```
<!-- comment format -->
```

Two hyphens at the beginning and end delineate the comment. It is a syntax error to use two or more hyphens within the body of the comment.

Generally, a command set contains one or more `loadContent` commands, other commands that manipulate the loaded content in some way, and one or more save commands that save the results.

When you load an input file, AGS determines the type of the file's contents and reads the file data into an appropriate content holder. AGS provides a number of different content holder types to hold the various file formats allowed as input. Once a file is loaded, it is called AGS *content*. Once you have loaded your input data, you can explicitly convert some content types to other content types, and perform other supported operations on the content.

When you save content, AGS outputs the content in one of its supported output file formats, depending on the type of content being saved. The act of saving content to one of the supported file formats may transform the content again, depending on a number of factors, such as the original input format, optimization settings in the content itself, and which save command you use to save it.

Content Holders and Types

When you load an input file, the content of the file being loaded is stored in a content holder. A content holder is temporary storage that AGS uses while it processes a request. This section explains that mechanism.

Almost all AGS commands create a new content holder to contain the result of the operation. Every content holder is of a particular *type*, which reflects the file type of the contents that go in it. The type of a content holder is determined by the command that creates it. See [“Content Types and File Formats” on page 43](#).

At any given time, there is one and only one *current* content holder. All other content holders are, by default, not current. You can think of the current status as a single label that moves around from content holder to content holder as command execution progresses. The next section explains this in more detail.

You can *name* a content holder with the `out` attribute of a command, so that you can access it in subsequent commands. (The AGS documentation generally refers to a named content holder simply as “named content.”) Most of the commands have one or more input attributes that take content holder names as values. If you do not name a content holder, you can only access it as long as it is current. When another content holder becomes current, the unnamed content holder is no longer accessible.

TIP: Try to order your commands to minimize the number of load operations for which you must name the resulting content. Doing this minimizes the amount of memory used during processing.

How Content Becomes Current

Content becomes current in two ways:

- The content holder created by a command to hold the result of that command becomes current when the command terminates. This is true whether or not you name the content with the `out` attribute.
- If you specify the `in` attribute for a command, the named content that you specify becomes current before the command performs its operation.

A command always operates on the current content. If you do not specify the `in` attribute for a command, it operates on the result of the previous command, regardless of whether that result was named.

You can reuse the same content names in multiple commands. When you assign a name to a new content holder, the content holder that previously had that name is discarded and becomes inaccessible.

If you use the same content name for both the `in` and `out` attributes of a command, the name is assigned to the result of the operation, and the content that was input to the command becomes inaccessible.

The save commands (`saveContent` and `saveOptimized`) write the current content to two places:

- *Into a new content holder* — This new content becomes current when the save command terminates, and the content is not transformed. You can optionally name the new content with the `out` attribute.
- *Into a named file* — The act of writing the current content to a file of a specific format may optimize the contents, if the content being saved is able to be optimized and optimization settings in the content are appropriately set.

How Content Holders Work

The following command sequence illustrates the changing state of content holders during execution, and is designed to illustrate all the important points about how content holders work. Each line in the sequence is numbered; the number corresponds to the explanation below of what happens when that line is executed, with an illustration of the content holder situation at that point. These commands load two PSD files and perform various rotation operations on them, saving them out to result files at the end.

```

1 <loadContent source="arrow.psd" out="image1" />
2 <loadContent source="square.psd" />
3 <rotate angle="45" out="image2turned" />
4 <rotate angle="90" />
5 <rotate in="image1" angle="180" out="image1turned" />
6 <saveContent name="pic1.psd" />
7 <saveContent in="image2turned" name="pic2.psd" />

```

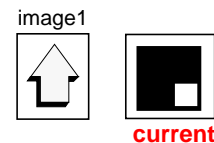
1. `<loadContent source="arrow.psd" out="image1" />`

The load command creates a content holder named `image1` into which it puts the contents of the input file `arrow.psd`. This becomes the current content holder when the command completes.



2. `<loadContent source="square.psd" />`

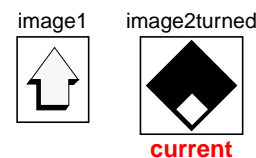
The load command creates a new unnamed content holder into which it puts the contents of the input file `square.psd`. This becomes the current content holder when the command completes.



3. `<rotate angle="45" out="image2turned" />`

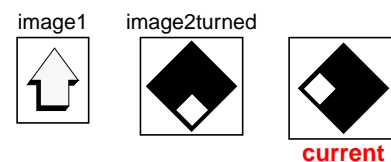
The rotate command operates on the current content. It creates another content holder named `image2turned` to hold the result of the rotate operation. This becomes the current content holder when the command completes.

Note that at this point, the content loaded in the second command is no longer accessible because that content holder was not named and is no longer current.



4. `<rotate angle="90" />`

The rotate command rotates the current content and puts the result in another, unnamed content holder. The new unnamed content becomes the current content when the command completes.

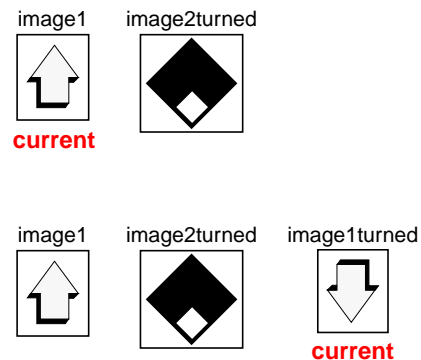


5. `<rotate in="image1" angle="180" out="image1turned" />`

This command resets the current content to a previously saved content holder, `image1`, (from the first load) before it performs its operation.

Note that at this point, the content produced by the fourth command is no longer accessible because the content holder it is in was not named and is no longer current.

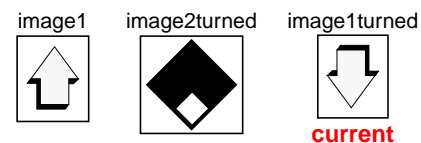
The command then rotates the current content, `image1`, 180 degrees and saves it in a new content holder, called `image1turned`, which becomes the current content.



6. `<saveContent name="pic1.psd" />`

This command creates a result file called `pic1.psd` from the current content.

When the command ends, `image1turned` is still the current content.

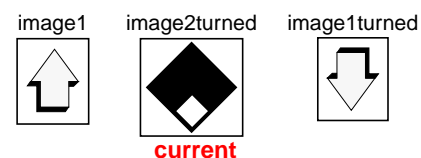


7. `<saveContent in="image2turned" name="pic2.psd" />`

This command resets the current content to the named content `image2turned`.

It then creates a result file called `pic2.psd` from the current content.

When the command ends, `image2turned` is still the current content.



Content Types and File Formats

[Table 3.1](#) lists all the allowed input file formats. For each input format, it shows what content type that type of input becomes, and what formats you could save that content in, *if you did not convert it to some other form before saving it or otherwise mark it to be saved in a different format*. Notice that you can save all types of content with the `saveContent` command, but only some types with the `saveOptimized` command. For image content types, the `saveOptimized` command determines the output format from optimization settings in the content itself. If the content does not specify optimization settings, the content is saved as GIF. For PDF content, the `saveOptimized` command saves it to PDF 1.4 format optimized for Web use.

TABLE 3.1 *How content types relate to input and output formats*

Input file format	Content type	Save command	Output file format
PSD	Raster	saveContent saveOptimized	PSD GIF, JPEG, PNG8, PNG24, WBMP
GIF	Raster	saveContent saveOptimized	PSD GIF, JPEG, PNG8, PNG24, WBMP
JPEG	Raster	saveContent saveOptimized	PSD GIF, JPEG, PNG8, PNG24, WBMP
PNG	Raster	saveContent saveOptimized	PSD GIF, JPEG, PNG8, PNG24, WBMP
TIFF	Raster	saveContent saveOptimized	TIFF GIF, JPEG, PNG8, PNG24, WBMP
SVG	SVG	saveContent saveOptimized	SVG GIF
PDF	PDF	saveContent saveOptimized	PDF Web-optimized PDF
PostScript	PostScript	saveContent	PostScript
EPS	PostScript	saveContent	EPS
Photoshop EPS	PostScript	saveContent	Photoshop EPS
XML	XML	saveContent	XML
XMP	XMP	saveContent	XMP

Loading Content

You must load all your input data into content holders before the AGS commands can access it. This section explains how you specify input data for a request, and how you load that data with the `loadContent` command in a command set. It also describes the details of how input file content may be transformed when it is loaded into content holders.

TIP: For best performance, minimize the number of input files in a request. For example, if you have six PSD files that you want to process in the same way, it is more efficient to send six requests, each with a short command set that loads and saves one file, than to send one request with a command set that loads and saves six files.

Specifying Input Data

If you send AGS a request using the Java, Perl, or COM API, you can include one or more input files with the request using the `addFile` method. You must explicitly load this input in your command set, using the name either you or AGS gave the input data. When you execute a request from the command line, you must also load input data explicitly, unless you are processing only one file and you specify that file on the command line.

Loading Input Data Explicitly

You use the `loadContent` command to explicitly load data into a content holder. If you submit a request with one of the APIs, the first command in your command set must be a `loadContent` command. If you submit a request from the command line, the first command in your command set must be a `loadContent` command unless you specify the `-source` argument on the command line.

The value of the `source` attribute is a file URI. The file it identifies must be locally accessible to the AGS process on the file system where it is running.

- If you invoke AGS from the command line, or you specify a command file on the server from one of the APIs (you use the `setCommandsFileOnServer` method on the `Request` object), relative URIs for the `source` attribute are resolved with respect to the location of the command file. You can always specify an absolute URI.
- If you invoke AGS from one of the APIs and include the commands in the `Request` object, the value you provide for the `source` attribute must be either an absolute file URI or the name associated with the content when the file was added to the request with the `addFile` command. These names cannot contain slashes or colons. A simple name directs AGS to get the input data from the `Request` object; an absolute URI directs it to the local file system.

TABLE 3.2 Specifying input data sources in the `loadContent` command

Form of source attribute	Details	Allowed
Absolute file URI: <code>file:///mydir/myfile.ext</code> (Solaris) <code>file:///C:/mydir/myfile.ext</code> (Windows)	Identifies a file on the local file system of the AGS process. Base input URI is ignored.	Always.
Absolute path: <code>/mydir/myfile.ext</code> <code>C:\mydir\myfile.ext</code>	Identifies a file on the local file system of the AGS process. Considered a relative URI, and resolved against the base input URI. In this case, only the scheme part of the base URI is needed to resolve the URI.	Allowed in all circumstances in which the command file exists on the local file system. This condition is always true when invoking AGS from the command line, and from the APIs when specifying the command file for the request with the <code>setCommandsFileOnServer</code> method.
Relative path: <code>mydir/myfile.ext</code> <code>myfile.ext</code> <code>myfile</code>	Identifies a file on the local file system of the AGS process. Considered a relative URI, and resolved against the base input URI. Allowed only when the command file resides on the local file system.	This condition is <i>not</i> true when invoking AGS from an API and including the command set in the request object (by using a <code>setFile*</code> command that doesn't end with "OnServer").
<i>name</i>	Identifies input data in the request object. In this case, the name must not contain slashes or colons (<code>\</code> , <code>/</code> , or <code>:</code>). The base input URI is the request object.	Required when invoking AGS from one of the APIs and referencing input data added to the request with the <code>addFile</code> method. If the request contains <i>any</i> input data added in this way, the only forms you can use for the source attribute are this one and an absolute URI.

Loading Input Data Automatically

Automatic loading occurs only when you invoke AGS from the command line. For files that you specify in command-line arguments, you can use either a relative or absolute path. In this case, relative paths are interpreted with respect to the current working directory.

- You can specify one input file on the command line with the `-source` argument. If you do this, AGS automatically loads that file into a content holder of the appropriate type named `srcfile`, before executing any commands in your command file.
- You can add variable replacement data on the command line, with the `-data` argument. If you do this, AGS automatically loads the replacement data into an XML content holder named `data`.

If you specify either or both of these arguments, you can still load other input data with explicit `loadContent` commands in your command file, and can reference the automatically loaded content by name in any command.

You can also specify user-defined variables and values on the command line. All variables and values are collected into an XML content holder named `commandLineArguments`.

For details of the command-line arguments, see the *Adobe Graphics Server API Reference*.

How File Formats Map to Content Types Upon Load

[Table 3.3](#) shows all the input file formats AGS can handle, and the type of content holder into which each format is read. The next section, [“Converting Content” on page 48](#), explains how you can transform loaded content to other content types.

AGS uses an internal, PSD-like format to store and operate on all raster image data. All raster-based file data is transformed to this format when loaded. The transformed image data is called *Raster content*. When Raster content is loaded, it is marked to be saved in a particular format. The section [“The Format Flag on Raster Content” on page 53](#) explains how you can change this setting.

In most cases, when you load an image file from PSD, TIFF, JPEG, or Photoshop EPS format that was produced by Photoshop, AGS reads Photoshop Image Resources such as clipping paths and color profiles and updates them as necessary. For more information on color handling for image formats, see [“Color Space Support in AGS” on page 94](#).

TABLE 3.3 Loading file data with the `loadContent` command

File format	Content type	Description	Typical uses
PSD	Raster	AGS automatically transforms the file contents to the internal Raster format, maintaining all layers and layer names. If the PSD file has a background layer, it is always the first layer, <code>layer[1]</code> . Any optimization settings that were present in the file are preserved. The content is marked to be saved in PSD format by the <code>saveContent</code> command.	Image data to be manipulated with the graphics commands.
GIF JPEG PNG	Raster	AGS automatically transforms the content to the internal Raster format with one unnamed layer, <code>layer[1]</code> . The content contains no initial optimization settings. The content is marked to be saved in PSD format by the <code>saveContent</code> command.	Image data to be manipulated with the graphics commands.

TABLE 3.3 Loading file data with the `loadContent` command (Continued)

File format	Content type	Description	Typical uses
TIFF	Raster	AGS automatically transforms the content to the internal Raster format with one unnamed layer <code>layer[1]</code> . The content contains no initial optimization settings. The content is marked to be saved in TIFF format by the <code>saveContent</code> command. If you subsequently add PSD features, such as new layers, to this content, and want to preserve them, you must explicitly mark the content to be saved as PSD before saving it.	Image data to be manipulated with the graphics commands.
SVG	SVG	AGS does not change SVG data when loading it. Any optimization settings are preserved.	Image data to be manipulated with the graphics commands.
PDF	PDF	AGS does not change PDF data when loading it.	Document input to be transformed to an image format.
PostScript EPS	PostScript	AGS does not change PostScript data when loading it. PostScript content is saved to the same format that was loaded.	Document input to be transformed to an image format.
XML	XML	AGS does not change XML data when loading it. You can load XML content in two different ways with the <code>loadContent</code> command: you can specify a file that contains the XML content with the <code>source</code> attribute, or you can omit the <code>source</code> attribute and provide the XML data inline within the <code>loadContent</code> element.	Replacement data for SVG content.
XMP	XMP	AGS does not change XMP data when loading it.	Replacement metadata values to be added to appropriate types of content.

Converting Content

Once you have loaded data into AGS with the `loadContent` command, you can manipulate it in a number of ways. The *Adobe Graphics Server Command Reference* contains reference information for all the AGS commands, and identifies which commands work on which type of content.

Some commands convert content of one type to content of another type. This section summarizes those conversions and points you to other locations in the documentation where these conversions are discussed in more detail.

Direct Content Type Conversions

AGS provides commands for converting some content types to other content types. All of the conversion commands take input content of one type, and output content of a different type, which becomes current. The following table summarizes the available content type conversions, and how to accomplish them.

TABLE 3.4 *Direct content type conversions*

Direct conversion		Command	Details
Raster	➡	EPS	<code>convertRasterToEPS</code>
	➡	PDF	<code>convertRasterToPDF</code>
SVG	➡	Raster	<code>convertSVGToraster</code> The Raster content is marked to be saved as PSD. Optimization settings that were set with the Illustrator Save For Web feature are preserved.
	➡	PDF	<code>convertSVGtoPDF</code>
PDF	➡	Raster	<code>convertPDFtoraster</code> The Raster content is marked to be saved as PSD, and contains no optimization settings.
PostScript EPS	➡	Raster	<code>convertPSToraster</code> The Raster content is marked to be saved as PSD, and contains no optimization settings.

You can perform these conversions in sequence to achieve a conversion for which there is no single explicit command. For example:

- To transform an SVG file to a PostScript file, load the SVG, convert it to PDF with the `convertSVGTtoPDF` command, convert the PDF content to EPS, and save with the `saveContent` command.
- To convert a PostScript file to a TIFF file, load the PostScript file, convert the PostScript content to Raster content, mark the Raster content to be saved as TIFF (see [“The Format Flag on Raster Content” on page 53](#)), and save with the `saveContent` command.
- To convert a PDF file to a JPEG file, load the PDF file, convert the PDF content to Raster content, use the `set` command to add the JPEG optimization settings, and save with the `saveOptimized` command.

Commands That Work on One Content Type but Produce Another

Most of the AGS commands, other than the conversion commands listed in [Table 3.4](#), produce the same type of content as the type on which they operate. A few commands, however, work on content of one type but produce content of a different type. These commands are listed in [Table 3.5](#).

TABLE 3.5 *Commands that work on one content type and produce another*

Command	Input content type	Output content type
imageInfo	Raster	XML
exportMetadata	Raster, SVG, PDF	XMP

Saving Content

The `saveContent` and `saveOptimized` commands save AGS content to an external file. You can call these commands to save content explicitly at any point in a command set. The contents may be saved to the local file system, or included in a response object if you have invoked AGS from one of the three APIs. Both save commands require the `name` attribute, which is used to name the file (on the file system) or result (in the response object).

If a command set does not contain an explicit save command, AGS automatically saves the current content after the last command has been processed; see [“Automatic File Saving” on page 55](#).

NOTE: Some files written out by AGS may not be binary compatible across different operating systems. For example, if you write a file on a Microsoft® Windows® system, it is not always identical to the same file written out on a Solaris system.

Determining Where Result Content is Saved

When you construct a request, you can tell AGS where to save the result files by specifying a *result location*. This must be a folder on the file system where the AGS process executes the request. You can specify the result location:

- *In the interface* — On the command line, give the `resultLocation` argument when you invoke AGS. From the Java, COM, or Perl API, call the `resultLocation` method when you construct the request.
- *In the commands* — Set the `resultLocation` attribute of the `commands` element.

NOTE: If you specify a result location in both the interface and the commands, the setting in the interface takes precedence.

Specifying the result location is optional. If you do not specify a result location, the default behavior for returning results is used. This default behavior varies across interfaces. The following table shows where results are placed in all cases:

TABLE 3.6 *Determining the result location*

Interface	Result location	Rule for determining destination
Java API Perl API COM API	specified	The name specified in the save command is interpreted as a file URI. If it is relative, the value of <code>resultLocation</code> is used as the base.
Web service direct	not specified	Content not saved with an absolute file: URI is returned to the client in a response object, identified by the name specified in the save command. The name must not be a file URI; it must not contain forward or backward slashes or a colon.
The command line	specified	The name specified in the save command is interpreted as a file URI. If it is relative, the value of <code>resultLocation</code> is used as the base.
	not specified	The name specified in the save command is interpreted as a file URI. If it is relative, the current working directory of the command line is used as the base URI.

Naming Results

If you have a command set that does not contain any save commands, AGS automatically saves the result of the last command. If the content type can be optimized, it uses the `saveOptimized` command; otherwise it uses the `saveContent` command. In either case, AGS names the result "Untitled" and appends an appropriate file extension.

When you explicitly save current content using a save command, you must specify a name for the result; both the `saveContent` and `saveOptimized` commands require the name attribute. The form this attribute value can take depends on how the request was invoked:

- If the request is invoked from one of the APIs and you have not specified a result location, the value of this name must be either an absolute URI (`file:///path`) or a simple name that does not contain slashes or colons (`myfile`). An absolute URI causes the result to be written to the local file system; a simple name causes it to be written to the response object.
- In all other cases, the value of the name can be a simple name, a name with a file extension, or a relative or absolute URI, including a filename. For requests submitted from the command line, relative URIs are resolved against the result location, if set, or against the location of the command file. For requests submitted from the APIs or directly to the Web service, relative URIs are allowed only if a result location has been specified, and are resolved relative to that location.

Providing Your Own File Extension

If you are sure of the file format being saved, and your results are being written to the file system, you can include an appropriate file extension as part of the name you supply. For example:

```
<saveContent name="pelican.svg" />
```

When you do this, AGS does not check that the extension you provide matches the actual format of the file written out.

Letting AGS Provide the File Extension

You can choose to have AGS automatically append the proper file extension. This guarantees the extension accurately reflects the file format. For example:

```
<saveContent name="pelican" appendExtension="true" />
```

If you set `appendExtension` to `true` and you also specify a file extension as part of the file name, you get a filename with two extensions. For example, if you save SVG content as follows:

```
<saveContent name="pelican.svg" appendExtension="true" />
```

the resulting file name is `pelican.svg.svg`.

Leaving Off File Extensions

You can also create a result with a name but no extension:

```
<saveContent name="pelican" />
```

If this command is submitted from one of the APIs, and the results are returned in the response object, you can use the methods in the appropriate Result class to extract both the name and the correct extension for the content.

Basic File Saving

The `saveContent` command saves AGS content to the external file system. The following table shows how the `saveContent` command transforms content to file formats.

TABLE 3.7 *Saving content with the `saveContent` command*

Content type	File format on save
Raster	Raster content is marked to be saved as PSD or TIFF. <ul style="list-style-type: none"> If it is marked as PSD, it is saved to a PSD version 7 file (<code>.psd</code>). If it is marked as TIFF, the <code>saveContent</code> command transforms the content to the TIFF format and saves it to a TIFF version 6 file (<code>.tif</code>).
SVG	Written as SVG (<code>.svg</code>), using the version and namespaces with which the content was loaded.
PDF	Written as PDF 1.4 (<code>.pdf</code>).

TABLE 3.7 *Saving content with the saveContent command (Continued)*

Content type	File format on save
PostScript	<p>Written to the format and version from which the content was loaded or created:</p> <ul style="list-style-type: none"> • PostScript (.ps) • encapsulated PostScript (.eps) • Photoshop-compatible encapsulated PostScript (.eps) <p>New PostScript files created with the <code>convertRasterToEPS</code> command are written as level 2 EPS.</p>
XML	Written as XML (.xml) using the version and namespaces with which the content was loaded.
XMP	Written as XMP (.xmp). For details of namespaces used in the file, see Chapter 4 , “Working with File Metadata.”

The Format Flag on Raster Content

A Raster content holder has a format flag that controls the format of the file produced by the `saveContent` command. When you load a PSD, JPEG, GIF, or PNG file, the flag is set to PSD. When you load a TIFF files, it is set to TIFF.

You can change the setting of this flag between loading and saving an image file, with the `setFileFormat` command. For example, you can load a GIF file and mark it to be saved as a TIFF file, or you can load a TIFF file and mark it to be saved as a PSD file. Similarly, you can convert another content type to Raster, then use `setFileFormat` to change the format flag, so that the content will be transformed to TIFF on save.

Metadata Conversions on Basic Save

The `saveContent` command saves all existing metadata with a file in the XMP format. For PSD and TIFF formats, applicable properties are also written in the EXIF and IPTC formats. Upon save, metadata values in all formats supported by AGS are guaranteed to be consistent.

If you do not want metadata saved with a file, you must explicitly remove it from the content before saving, using the `removeMetadata` or `importMetadata` command. For more information, see [Chapter 4](#), “Working with File Metadata.”

Saving to Optimized Formats

You can save Raster, SVG, or PDF content to a Web-optimized file format with the `saveOptimized` command. The command saves to optimized formats depending on the current content type, as follows:

TABLE 3.8 *Saving content with the `saveOptimized` command*

Content type	Save behavior
Raster	<p>The <code>saveOptimized</code> command saves to one of the optimized image formats — GIF, JPEG, PNG8, PNG24, or WBMP — depending on the optimization settings in the content.</p> <p>The <code>saveOptimized</code> command flattens layers in the current content before creating the result file according to the current optimization settings. AGS uses optimization settings that were set in Photoshop using Save For Web, or set previously in AGS. You can use the <code>optimizeToSize</code> and <code>set</code> commands to modify optimization settings within AGS. Any settings that are not explicitly provided default according to the preferred file format.</p> <p>If the current content does not contain optimization settings, the command writes a GIF file using built-in default optimization settings.</p> <p>The command operates only on images that use RGB color profiles. You can use the <code>convertProfile</code> command to convert the color to an RGB profile.</p>
SVG	<p>The <code>saveOptimized</code> command converts SVG content to the Raster format, then saves it in the preferred format using any optimization settings that were set in Illustrator using Save For Web, or set previously in AGS. You can use the <code>set</code> command to modify optimization settings within AGS.</p> <p>If the current content does not contain optimization settings, the command writes a GIF file using built-in default optimization settings.</p>
PDF	<p>The <code>saveOptimized</code> command saves to PDF 1.4 format optimized for Web use (the "Fast Web View"). This may make it much faster to read the file from a Web site.</p> <p>You cannot set any specific optimization parameters for PDF content.</p>

The following table summarizes the file name extensions that this command assigns for file formats, and the versions of those formats that it writes.

TABLE 3.9 *File extensions used by the `saveOptimized` command*

Format	Extension	Version
GIF	.gif	Written as version 89a.
JPEG	.jpeg	Written as JFIF 1.2.
PNG8/24	.png	Written according to the PNG 1.2 specification (the PNG file format itself is not versioned).
WBMP	.wbmp	Written as a WAP 1.1 type 0 image.
PDF	.pdf	Written as linearized PDF 1.4.

For more information, see [“Saving Optimized Files” on page 104](#).

Setting Optimization Information

There are two ways you can set or modify the optimization information in content before saving it:

- Use the `optimizeToSize` command to automatically set the preferred optimized file format for an RGB image. This command determines whether GIF or JPEG format results in a smaller file.
- Use the `set` command to explicitly modify or replace the `optimizationSettings` element in the content. The optimized save format is determined by the name of the child element you provide. The choices are:
 - `GIFFormat`
 - `PNG8Format`
 - `PNG24Format`
 - `WBMPFormat`
 - `JPEGFormat`

The default optimization file format is GIF.

Saving Metadata with Optimized Files

In the `saveOptimized` command, you can use the optional `metadata` attribute to specify whether to save metadata with an optimized image file, and what metadata formats to save (XMP, EXIF, and/or IPTC). If you do not supply the attribute, no metadata is saved with the image file. (XMP metadata is always saved with PDF files.)

You can specify any combination of the supported formats; however, the non-XMP formats apply only to JPEG files. If the file is not being written in JPEG format, those values are ignored. You can include XMP format metadata for any output file.

If you specify any non-XMP formats for a JPEG file, the properties appropriate to the format are saved with the file. If you also specify XMP format, all current XMP metadata is saved with the file. Upon save, metadata values in all formats are guaranteed to be consistent.

For more information, see [Chapter 4, “Working with File Metadata.”](#)

Automatic File Saving

After every command in the request has been processed, in the absence of any explicit `saveOptimized` or `saveContent` commands, AGS performs an implicit save operation on the current content. Which save command it uses depends on the type of the current content.

Content type	Save command used
RGB Raster, SVG, PDF	<code>saveOptimized</code>
CMYK Raster, PostScript, XML, XMP	<code>saveContent</code>

When a result file is created by an automatic save operation, it is given the name `Untitled`, and the appropriate file extension is appended.

When you pass a file to AGS on the command line in the `-source` argument and do not specify a command file, an automatic save is performed on the file, and the result is saved with the default name in the current working directory or the result location, if specified.

Referenced Attributes Values

AGS defines a special reference function you can use to access named content and retrieve values from it, as long as that content is accessible with abbreviated XPath syntax. You can use such a content reference anywhere as an attribute value. A content reference can be useful for retrieving values from loaded content at run time, when you do not know ahead of time what those values will be.

When AGS executes a command containing a content reference, it first evaluates the reference, then passes the result to the command as the attribute value. If the reference does not evaluate to the expected value type, the command reports an error.

The syntax for a content reference is as follows:

```
reference(adobe-content:contentName#XPath)
```

The `#XPath` portion of the specification is optional. If it is not supplied, the expression evaluates to the entire named content.

For example, suppose you load and name the following simple XML content:

```
<loadContent out="myXml">
  <A><B d="dog" /></A>
</loadContent>
```

- The following reference extracts the string value "dog" from the `d` attribute of the `B` element:

```
reference(adobe-content:myXml#/A/B/@d)
```

You could pass this reference for an attribute that expected a text-string value.

- The following reference, with no XPath, evaluates to the entire named content:

```
reference(adobe-content:myXml)
```

Note that this reference does *not* evaluate to the content-name string, "myXml", but to the entire named content, the string "`<A><B d='dog' />`".

EXAMPLE 3.1 Using a content reference to select an image at run time

The following commands load the sample XML shown above from an external source and use the reference syntax to select and operate on the dog image:

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="dog.psd" out="dog" />
  <loadContent source="cat.psd" out="cat" />
  <loadContent source="selectPet.xml" out="myXml" />
  <flip in="reference(adobe-content:myXml#/A/B/@d)"
        direction="vertical" />
  <saveContent name="upsideDownPet.psd" />
</commands>
```

In this example, you do not need to know which of the loaded images is wanted when you write the commands. The name of the desired image file is specified in the `selectPet.xml` file; the content of this file is accessed during execution to determine which content name is used by the `flip` command.

General Commands

The following commands report on or affect the state of the system and do not operate on content:

- `info`: Reports information about the AGS application, such as copyright information and Adobe personnel involved on the project, in XML format.
- `version`: Reports the current AGS version information in XML format.
- `registerMetadataNamespace`: Creates an XML namespace for use with metadata, and requests a prefix to be associated with it. For more information, see [Chapter 4, “Working with File Metadata.”](#)

4

Working with File Metadata

This chapter describes how AGS deals with metadata in the files that you can load and manipulate. It contains the following sections:

Overview of Metadata	starting on page 59
Working with Metadata in AGS	starting on page 61
Transformations of Metadata Formats	starting on page 69

Overview of Metadata

AGS uses Extensible Metadata Platform technology (XMP) to handle metadata associated with many content types. XMP is a framework for XML metadata exchange and a labeling system for document and digital assets. XMP defines a mechanism for placing metadata in a variety of file formats.

AGS can read and write metadata values in both XMP (for Raster, SVG, and PDF content) and in the EXIF and IPTC formats (defined for PSD, TIFF, and JPEG). However, it manages all metadata internally in the XMP format, and you must use the XMP format for modifying metadata. AGS handles the translation to and from XMP and other formats upon load and save; see [“Transformations of Metadata Formats” on page 69](#). When you load a file into AGS, AGS resolves any conflicts in corresponding metadata values from different formats. When you save a file from AGS, metadata values are guaranteed to be consistent in all formats.

AGS operates only on document-level metadata. If a file contains component-level metadata (such as metadata stored in an embedded JPG image in a PDF file), AGS preserves it, but does not allow you to access it.

AGS supplies commands for extracting and replacing metadata, and for manipulating individual metadata values. See [“Working with Metadata in AGS” on page 61](#).

Metadata and Content Types

AGS represents and manipulates metadata in the XMP format. When it loads a file, it translates all supported metadata formats into the XMP format. Properties from each original format go into a specific namespace, so that they can be restored to that format on save. See [“Transformations of Metadata Formats” on page 69](#).

You can extract metadata from a file using the `exportMetadata` command. The metadata is returned as XMP content. XMP content contains data in the XML Packet format; however, it is not the same as XML content. XMP and XML content cannot be used interchangeably.

- When you save the XMP content, the `saveContent` command provides the `.xmp` file extension and saves the file as an XML Packet using the XMP packet wrapper.
- You can reload the saved metadata in another AGS session using the `loadContent` command, and, if you wish, use it to replace all of the metadata for a file, using the `importMetadata` command. When AGS loads an XML Packet with the XMP packet wrapper, it treats it as XMP content, rather than XML content.

All metadata manipulation operations other than `importMetadata` act on XMP metadata embedded in content of the various supported types: PDF, Raster, and SVG.

The `saveContent` command automatically transforms the current XMP metadata to all metadata formats supported for the content type, and embeds it in the output file. The `saveOptimized` command allows you to specify whether or not to write out metadata with the optimized file, and, if you include metadata, which formats to include.

Creating a Custom Metadata Schema

AGS defines standard XMP namespaces, and in addition defines namespaces for XMP properties that are created from EXIF- and IPTC-format metadata. See [“Metadata Namespaces” on page 71](#). If you add or modify properties in these namespaces, the `save` commands can write out that information to the EXIF and IPTC formats. See [“Saving Files with Metadata” on page 68](#).

Within AGS, you can register your own metadata namespace that identifies a custom metadata scheme. Before manipulating the metadata, register your namespace with AGS using the `registerMetadataNamespace` command. The namespace is then recognized throughout the current AGS request, and you can add or modify properties in that namespace, using the metadata manipulation commands.

When AGS writes out XMP metadata, by itself or embedded in a file, each namespace is associated with a prefix in the XML Packet. For example, the following XML Packet fragment specifies that the namespace `http://ns.adobe.com/photoshop/1.0/` is associated with the prefix `photoshop`, as in the following code fragment:

```
<rdf:Description about=''
  xmlns:photoshop='http://ns.adobe.com/photoshop/1.0/'>
<photoshop:Author>jsmith</photoshop:Author>
```

Thereafter, the file always uses the prefix to refer to that namespace (as illustrated here for the `Author` property).

In the `registerMetadataNamespace` command, you suggest a prefix to be used for your namespace. If that prefix is already in use, AGS assigns a unique generated prefix for the namespace.

You must always use a complete namespace to specify metadata properties in AGS metadata manipulation commands.

Working with Metadata in AGS

When you load a file into AGS, it creates an XMP representation of that file's metadata, translating properties as needed from IPTC and EXIF metadata formats.

- You can extract metadata from a loaded file and write it out to an XMP file.
- You can modify the metadata in a loaded file by adding or deleting properties, changing property values, or completely replacing the metadata with a new XML Packet.

Extracting Metadata from Loaded Content

To extract metadata in AGS content, use the `exportMetadata` command. This command copies all of the existing XMP-format document-level metadata for the current content into a new XMP content holder, which contains the data in the XML Packet format. The command makes the new XMP content current. The XMP contains the namespaces required to preserve metadata that was loaded from other supported formats (IPTC and EXIF), as well any other namespaces used in the file.

Because AGS does not support program-logic branching in its command syntax, you cannot make metadata-based decisions inside AGS. You can, however, extract and write out metadata to a separate file, read that file outside AGS, and make decisions based on the values.

Once you have extracted metadata from AGS content, you can use the `saveContent` command to save the extracted XMP data as an XML Packet. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="myImage.psd" />
  <flatten out="flatImg"/>
  <exportMetadata />
  <saveContent name="myImageMetadata" appendExtension="true" />
  <saveContent in="flatImg" name="flatImage" appendExtension="true" />
</commands>
```

This example creates a file `myImageMetadata.xmp`, which contains:

- All of the XMP metadata loaded with `myImage.psd`.
- Any properties that were originally defined in the IPTC and EXIF formats, in the appropriate namespaces.
- Metadata properties created by AGS to reflect its own manipulation of the file (see [“Automatic Metadata Updates” on page 73](#)).

EXAMPLE 4.1 Metadata from IPTC format

The following example shows an XML Packet containing metadata returned by this process, from a PSD file in which much of the original metadata was in the IPTC format:

```
<?xpacket begin='i>?' id='W5M0MpCehiHzreSzNTczkc9d'?>
<?adobe-xap-filters esc="CRLF"?>
<x:xapmeta xmlns:x='adobe:ns:meta/' x:xaptk='XMP toolkit 2.8.2-33,
  framework 1.5'>
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:iX='http://ns.adobe.com/iX/1.0/'>
<rdf:Description about=''
  xmlns:photoshop='http://ns.adobe.com/photoshop/1.0/'>
  <photoshop:Author>jsmith</photoshop:Author>
  <photoshop:AuthorsPosition>photographer</photoshop:AuthorsPosition>
  <photoshop:Caption>calvin on a boat</photoshop:Caption>
  <photoshop:Category>cat</photoshop:Category>
  <photoshop:City>Minneapolis</photoshop:City>
  <photoshop:Copyright>do not copy this</photoshop:Copyright>
  <photoshop:Country>USA</photoshop:Country>
  <photoshop:ObjectName>calvin</photoshop:ObjectName>
  <photoshop:TransmissionReference>somewhere
    </photoshop:TransmissionReference>
  <photoshop:SupplementalCategories>
    <rdf:Bag>
      <rdf:li>cat4</rdf:li>
      <rdf:li>cat5</rdf:li>
    </rdf:Bag>
  </photoshop:SupplementalCategories>
</rdf:Description>
</rdf:RDF>
</x:xapmeta>
<?xpacket end='r'?>
```

EXAMPLE 4.2 Metadata from EXIF format

The following is an example of the XMP representation of metadata that was originally in the EXIF format:

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<rdf:Description about='' xmlns:tiff='http://ns.adobe.com/tiff/1.0'
  tiff:Make='Canon'
  tiff:Model='Canon PowerShot S300'
  tiff:Orientation='1'
  tiff:XResolution='180'
  tiff:YResolution='180'
  tiff:ResolutionUnit='2'
  tiff:DateTime='2001-07-25T20:18:27Z'
  tiff:YCbCrPositioning='1'>
</rdf:Description>
<rdf:Description about='' xmlns:exif='http://ns.adobe.com/exif/1.0'
  exif:ExposureTime='167/10000'
  exif:FNumber='27/10'
  exif:ExifVersion='0210'
  exif:DateTimeOriginal='2001-07-25T20:18:27Z'
  exif:DateTimeDigitized='2001-07-25T20:18:27Z'
  exif:CompressedBitsPerPixel='3'
  exif:ShutterSpeedValue='59/10'
  exif:ApertureValue='9/10'
  exif:ExposureBiasValue='0'
  exif:MaxApertureValue='28659/10000'
  exif:SubjectDistance='913/1000'
  exif:MeteringMode='5'
  exif:Flash='1'
  exif:FocalLength='541/100'>
  <exif:ComponentsConfiguration>
    <rdf:Seq>
      <rdf:li>1</rdf:li>
      <rdf:li>2</rdf:li>
      <rdf:li>3</rdf:li>
      <rdf:li>0</rdf:li>
    </rdf:Seq>
  </exif:ComponentsConfiguration>
<!-- exif:ImageDescription is aliased to dc:title -->
</rdf:Description>
<rdf:Description about='' xmlns:dc='http://purl.org/dc/elements/1.1/'>
<dc:title>
  <rdf:Alt>
    <rdf:li xml:lang='x-default'>The image's title</rdf:li>
  </rdf:Alt>
</dc:title>
</rdf:Description>
</rdf:RDF>
```

EXAMPLE 4.3 XMP Metadata in a PDF file

The following is an example of XMP metadata extracted from PDF content:

```
<?xpacket begin='?' id='W5M0MpCehiHzreSzNTczkc9d'?>
<?adobe-xap-filters esc="CRLF"?>
<x:xapmeta xmlns:x="adobe:ns:meta/"
  x:xaptk="XMP toolkit 2.8.2-33, framework 1.5">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  " xmlns:ix="http://ns.adobe.com/ix/1.0/">
<rdf:Description about="" xmlns:pdf="http://ns.adobe.com/pdf/1.3/">
  <pdf:Producer>Acrobat Distiller 5.0.5 (Windows)</pdf:Producer>
  <!-- pdf:CreationDate is aliased -->
  <!-- pdf:ModDate is aliased -->
  <!-- pdf:Creator is aliased -->
  <!-- pdf:Author is aliased -->
  <!-- pdf:Title is aliased -->
</rdf:Description>
<rdf:Description about="" xmlns:xap="http://ns.adobe.com/xap/1.0/">
  <xap:CreateDate>1997-04-15T18:17:39Z</xap:CreateDate>
  <xap:ModifyDate>2002-07-25T13:58:04-05:00</xap:ModifyDate>
  <xap:MetadataDate>2002-07-25T13:58:04-05:00</xap:MetadataDate>
  <xap:CreatorTool>FrameMaker 6.0</xap:CreatorTool>
  <!-- xap:Authors is aliased -->
  <!-- xap:Author is aliased -->
  <!-- xap:Title is aliased -->
</rdf:Description>
<rdf:Description about="" xmlns:xapTPg="http://ns.adobe.com/xap/1.0/t/pg/">
  <xapTPg:NPages>23</xapTPg:NPages>
</rdf:Description>
<rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:creator>
    <rdf:Seq>
      <rdf:li>jsmith</rdf:li>
    </rdf:Seq>
  </dc:creator>
  <dc:title>
    <rdf:Alt>
      <rdf:li xml:lang="x-default">Sample.fm</rdf:li>
    </rdf:Alt>
  </dc:title>
</rdf:Description>
</rdf:RDF>
</x:xapmeta>
<?xpacket end='r'?>
```

Modifying Metadata in Loaded Content

You can modify the metadata in a loaded file by adding or deleting properties, changing property values, or completely replacing the metadata with a new XML Packet.

Adding and Modifying Non-Structured Properties

To add a new non-structured (scalar) property and its value, or to change the value of an existing property, use the command `setMetadata`.

When you add new metadata in AGS using the `setMetadata` command, it is created according to the W3C Resource Description Framework (RDF). For more information on this syntax, see <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>.

AGS uses a set of aliases defined in the XMP specification for equivalent properties. Any alias can be used to refer to the source property, which contains the value. When you specify an alias as the property to be modified, AGS modifies the source. (See “[Metadata Aliases](#)” on [page 72](#).)

EXAMPLE 4.4 Modifying a metadata property

This example loads a PSD file, sets the metadata property `AuthorsPosition` to the value `artist`, then saves the content to a new PSD file. The file is saved with embedded XMP, EXIF, and IPTC metadata. The new property value is in the namespace for IPTC, so the new value will be seen both in the XMP data and in the IPTC metadata that you can view in Photoshop, as `<photoshop:AuthorsPosition>artist</photoshop:AuthorsPosition>`.

Note that you must supply the complete namespace in the namespace property, not the prefix.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="myImage.psd" />
  <setMetadata namespace="http://ns.adobe.com/photoshop/1.0/"
    path="AuthorsPosition"
    value="artist" />
  <saveContent name="newImage" appendExtension="true" />
</commands>
```

Adding and Modifying Structured Properties

Structured metadata properties are containers for multiple items. There are three types of structured properties, which reflect different restraints on the value set:

- The type *bag* contains an unordered list of items.
- The type *sequence* contains an ordered sequence of items.
- The type *alternative* contains a set of alternative items.

Use the command `createMetadata` to create a new structured property along with its first item. For an existing structured property, use the command `appendMetadata` to add a new item, before or after an existing item. Use the command `setMetadata` to set the value of an existing item.

EXAMPLE 4.5 Creating and adding to a structured property

This example loads a PSD file and creates a new structured metadata property named `Keywords` in the standard XMP namespace. The new property is of type `bag`, indicating an unordered list of items. The first item is `Keyword1`. The example then adds a second item to the property, `Keyword2`, appending it to the first item (regardless of what that item was). It then saves the Raster content as a new PSD file (with embedded XMP, IPTC, and EXIF metadata), changes the value of the second `Keywords` item, and saves the file with the new value.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="cat.psd" />
  <createMetadata namespace="http://ns.adobe.com/xap/1.0/"
    path="Keywords" value="Keyword1" type="bag" />
  <appendMetadata namespace="http://ns.adobe.com/xap/1.0/"
    path="Keywords/*[1]" value="Keyword2" out="cat" />
  <saveContent name="catWkeys" appendExtension="true" />
  <setMetadata in="cat" namespace="http://ns.adobe.com/xap/1.0/"
    path="Keywords/*[2]" value="newKeyword" />
  <saveContent name="catNewkeys" appendExtension="true" />
</commands>
```

Deleting Metadata

You can delete an individual property, or delete all current metadata by replacing it with a different XMP data. To delete an individual property, use the `removeMetadata` command, specifying the full namespace and name of the property. For example:

```
<removeMetadata namespace="http://ns.adobe.com/pdf/1.3/" path="Author" />
```

To remove an item from a structured property, provide the XPath to that item in the `path` attribute. For example:

```
<removeMetadata namespace="http://ns.adobe.com/xap/1.0/"
  path="Keywords/*[2]" />
```

AGS uses a set of aliases defined in the XMP specification for equivalent properties. You can use any defined alias to refer to the source property, which contains the value. When you specify an alias as the property to be deleted, AGS deletes the source property. (See [“Metadata Aliases”](#) on page 72.)

- If you do not specify any namespace in the `removeMetadata` command, AGS deletes all of the current metadata and replaces it with a default XML Packet containing those properties automatically updated by AGS.
- If you specify the namespace but not the path, AGS deletes all metadata in the specified namespace.

Replacing Metadata

You can completely replace the metadata for the current content with a new set of properties, with the `importMetadata` command. All metadata associated with the content is removed and replaced with the new XML Packet that you provide.

You can specify the new XML Packet in one of several ways, as shown in the next three examples.

EXAMPLE 4.6 Streaming in replacement metadata in the `importMetadata` command

This example loads the XML Packet inline between `importMetadata` tags.

NOTE: You cannot load XMP data inline into AGS using `loadContent`. The `loadContent` command would interpret the XML Packet as XML and store it as XML content, which you cannot pass to `importMetadata`.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="myFile.pdf" />
  <importMetadata>
    <rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
      xmlns:ix='http://ns.adobe.com/ix/1.0/'>
      <rdf:Description about=''
        xmlns='http://ns.adobe.com/pdf/1.3/'
        xmlns:photoshop='http://ns.adobe.com/photoshop/1.0/'>
        <photoshop:DateCreated>2002-08-28T09:08:09Z</photoshop:DateCreated>
      </rdf:Description>
    </rdf:RDF>
  </importMetadata>
  <saveContent name="myFile-newMdata.pdf" />
</commands>
```

EXAMPLE 4.7 Loading replacement metadata from a file

This example loads the XML Packet from an XMP file and names it using the `out` attribute of `loadContent`. It is stored as XMP content. You then refer to the named XMP content in the `source` attribute of `importMetadata`.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="myFile.pdf" out="tobeupdated" />
  <loadContent source="mymetadata.xmp" out="md" />
  <importMetadata in="tobedupated" source="md" />
  <saveContent name="myImg-new-mdata.pdf" />
</commands>
```

EXAMPLE 4.8 Copying metadata from one file to another

This example extracts and stores metadata from one PSD file, then loads another PSD file. It refers to the stored XMP content as the replacement value for metadata in the new current content.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="myImage1.psd" />
  <exportMetadata out="mdata" />
  <loadContent source="myImage2.psd" />
  <importMetadata source="mdata" />
  <saveContent name="myImg2-new-mdata.psd" />
</commands>
```

Saving Files with Metadata

When you save a file with the `saveContent` command, AGS writes out all of the current metadata with the file in the XMP format. If there was any metadata transformed from other formats on load, it is copied back to that format on save. Because the load process resolved all conflicts between different formats, metadata values are guaranteed to be consistent in all formats when you save a file from AGS.

AGS derives some metadata values automatically (see [“Automatic Metadata Updates” on page 73](#)). These values are saved along with values that have been preserved from load or explicitly set in AGS. If you do not want to save a metadata property with a file, use the `removeMetadata` command to remove it before saving the file. However, you cannot remove those metadata properties that are automatically updated by AGS.

Specifying Formats for Optimized Image Files

When you save optimized image files, you can specify whether to save metadata with the files, and which formats to save it in. To do this, use the `metadata` attribute of the `saveOptimized` command.

The `metadata` attribute takes a list of supported metadata formats to save, separated by spaces. You can give any combination of the three formats, or none. If you do not supply the attribute, or if you give an empty string, the command does not save any metadata with the file.

The three supported formats are XMP, EXIF, and IPTC. Of the optimized file types, only JPEG files can be written with EXIF and/or IPTC metadata. If you specify a format that is not supported for the type of file being saved, that attribute value is ignored.

The optimization settings for the JPEG file format also include an `embedMetadata` attribute. (See the chapter “Accessing Raster Content” in the *Adobe Graphics Server XML Grammars Reference*.) If this attribute is `true`, IPTC metadata is embedded in the file regardless of the value of the `saveOptimized metadata` attribute. In this case, you can use the `saveOptimized metadata` attribute to specify XMP and/or EXIF in addition to the IPTC format.

EXAMPLE 4.9 Saving an optimized image with a new metadata property

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="cat.jpg" out="cat" />
  <setMetadata namespace="http://ns.adobe.com/tiff/1.0/" path="Artist"
    value="jsmith" />
  <set target="/psd/optimizationSettings" >
    <optimizationSettings>
      <JPEGFormat embedMetadata="true" />
    </optimizationSettings>
  </set>
  <saveOptimized name="newCat.jpg" metadata="xmp exif" />
</commands>
```

Transformations of Metadata Formats

When you load files into AGS, AGS uses a set of aliases and precedence rules to resolve equivalent metadata properties from different formats to a single value. It automatically derives and adds certain metadata values that reflect its own manipulation of the files.

When you save content containing metadata, the `saveContent` command transforms the information into all of the metadata formats that apply to the file format, and saves them along with the XMP. This ensures that metadata within and saved from AGS is consistent across all formats. The `saveOptimized` command similarly transforms the metadata into those formats that you specify, if they apply to the saved file format.

This section describes:

- Metadata formats that AGS supports for each file type.
- The namespaces AGS uses to represent non-XMP metadata.
- The aliases that define equivalencies among namespaces.
- The metadata values that AGS derives automatically.
- The precedence rules AGS uses to resolve property value conflicts.

Supported Metadata Formats

[Table 4.1](#) shows the metadata formats that AGS supports for each file format. It describes how AGS treats the metadata in each kind of file upon load, and how AGS treats metadata when content loaded from one format is converted to a different format.

TABLE 4.1 **Supported metadata formats**

File format	Metadata formats	Metadata support on load, save, and convert
PSD	XMP, IPTC, EXIF	<p>AGS preserves EXIF, IPTC and XMP metadata on load.</p> <ul style="list-style-type: none"> The <code>saveContent</code> command writes out all supported metadata formats for PSD and TIFF files. The <code>saveOptimized</code> command writes out only the metadata formats you specify that are supported for the destination file format. Only the JPEG format supports EXIF and IPTC formats. The optimization settings for JPEG also affect whether IPTC metadata is saved with the file.
JPEG	XMP, IPTC, EXIF	
TIFF	XMP, IPTC, EXIF	
GIF	XMP	
PNG	XMP	
SVG	XMP	<p>AGS preserves the document-level XMP metadata in SVG files on load, but ignores metadata stored in other metadata elements.</p> <p>When you convert SVG content to PDF content, some of the PDF metadata is updated (see “Automatic Metadata Updates” on page 73). Metadata embedded in images that are referenced in the SVG remains inside the image in the resulting PDF file, but is not accessible to AGS.</p>
PDF	XMP, PDF 1.4	<p>The rules for XMP metadata propagation to the PDF Info dictionary are described in the XMP Framework documentation.</p> <p>Upon load, AGS updates the <code>NPages</code> property in the namespace <code>http://ns.adobe.com/xap/1.0/t/pg/</code> to reflect the number of pages in the document.</p> <p>When you use the <code>convertPDFToRaster</code> command, the document-level metadata is preserved, but any metadata associated with embedded images is lost.</p> <p>For operations that combine multiple PDF documents, or multiple portions of PDF documents, the metadata of the destination document is preserved, and the metadata for the donor document is discarded. Metadata carried inside images in both documents is preserved. If necessary, you can retrieve desired metadata before the conversion operation, then restore it after the operation.</p>
PhotoShop EPS	XMP	<p>When you convert an EPS file created by Photoshop to Raster format, XMP metadata is preserved and can be saved with the file.</p> <p>When you convert Raster content to Photoshop EPS, all metadata is preserved in XMP format and is saved by the <code>saveContent</code> command.</p>

TABLE 4.1 Supported metadata formats (Continued)

File format	Metadata formats	Metadata support on load, save, and convert
generic EPS, PostScript	--	<p>AGS cannot access metadata in generic EPS or PostScript files, even when the metadata is stored in XMP format. Any metadata that was in the loaded file is saved with the file by the <code>saveContent</code> command.</p> <p>When you convert EPS files that were not created by Photoshop to Raster content, AGS does not preserve metadata, even if it is in XMP format.</p>

Metadata Namespaces

AGS uses the following preregistered XMP namespaces:

TABLE 4.2 Preregistered namespaces

Namespace	Prefix
http://ns.adobe.com/xap/1.0/	xap
http://ns.adobe.com/xap/1.0/g/	xapG
http://ns.adobe.com/xap/1.0/g/img/	xapGImg
http://ns.adobe.com/xap/1.0/dyn/	xapDyn
http://ns.adobe.com/xap/1.0/dyn/a/	xapDynA
http://ns.adobe.com/xap/1.0/dyn/v/	xapDynV
http://ns.adobe.com/xap/1.0/t/	xapT
http://ns.adobe.com/xap/1.0/t/pg/	xapTPg
http://ns.adobe.com/xap/1.0/rights/	xapRights
http://ns.adobe.com/xap/1.0/mm/	xapMM
http://ns.adobe.com/xap/1.0/s/	xapS
http://ns.adobe.com/xap/1.0/bj/	xapBJ
http://ns.adobe.com/pdf/1.3/	pdf
http://purl.org/dc/elements/1.1/	dc
http://ns.adobe.com/tiff/1.0	tiff
http://ns.adobe.com/exif/1.0	exif
http://ns.adobe.com/photoshop/1.0/	photoshop

When you load a file containing metadata into AGS, AGS transforms properties from EXIF and IPTC formats into XMP equivalents in their respective namespaces. The properties can be transformed back to the original format upon save.

- Properties read from the EXIF format may be in the `tiff` or `exif` namespace.
AGS base-64-encodes some EXIF values. In these cases, the value is prefaced with the string `'base64,'`.
- Properties in the `photoshop` namespace are equivalent to fields available through the PhotoShop File Info dialog, which PhotoShop may store as IPTC or XMP or both.
Because IPTC metadata is often stored without any encoding information, AGS uses the following rules to map unencoded character values when converting between IPTC and XMP:
 - Invalid 7-bit characters appearing in IPTC fields (that is, non-graphic characters other than space, newline, carriage return, and tab) are mapped to a question mark when converted to XMP.
 - 8-bit characters appearing in IPTC fields without encoding information are mapped to a question mark when converted to XMP. (IPTC allows 8-bit encodings to be specified, but Photoshop does not.)
 - Non-ASCII characters in XMP fields are converted to an ASCII character (question mark or appropriate fallback character) when mapping from XMP to IPTC.

Metadata Aliases

To ensure consistency of metadata, AGS uses equivalencies among the metadata formats that it supports. The XMP toolkit defines aliases from equivalent properties in other namespaces to a source property, usually in the Dublin Core (`dc`) namespace. When AGS reads in metadata from an aliased property, it stores the value in the source property.

You can refer to a property by any of its aliases. When you specify an alias in a metadata command, AGS performs the operation on the source property for that alias.

For example, the XMP property `dc:description` is the source for the aliased fields `xap:Description`, `tiff:imageDescription`, and `photoshop:Caption`. When AGS reads a file that has any one of these values, it creates the source property (`dc:description`). The source property contains the value, regardless of where the value came from. The aliases do not have values, but simply point to the source, `dc:description`.

The following fragment of an XML Packet illustrates how aliases are shown at the beginning of the packet:

```
<rdf:Description about='uuid:1ad2dd29-1ce1-11d6-b461-a6fbf894a6ca'
xmlns:pdf='http://ns.adobe.com/pdf/1.3/'>
<!-- pdf:Subject is aliased -->
<!-- pdf:Author is aliased -->
<!-- pdf:Title is aliased -->
</rdf:Description>
```

Automatic Metadata Updates

When you save content from AGS, some internal metadata properties such as `xap:MetadataDate` are automatically updated. Other metadata values, such as the size or bit depth of an image, that are derived from the data, cannot be set explicitly, but may be updated automatically on save.

Derived properties are not automatically copied to the IPTC or EXIF metadata formats. If metadata is not present in either IPTC or EXIF format in the source document, it is not present in the output in that format, unless you specifically add it using AGS commands.

The following table lists metadata properties that are automatically updated during format conversion and file save operations.

TABLE 4.3 *Metadata fields modified by AGS*

Command	Fields modified
saveContent (SVG)	xap:Format xap:MetaDate xap:ModifyDate
saveContent (PDF)	pdf:ModDate xap:Format xap:MetaDate xap:ModifyDate
saveContent (EPS)	xap:Format xap:MetaDate xap:ModifyDate NOTE: Metadata is not saved with PostScript content that does not conform to PhotoShop-compatible encapsulated PostScript.
saveContent (PSD)	stDim:w
saveOptimized (JPG, GIF, PNG)	stDim:h xap:Format xap:MetaDate xap:ModifyDate
saveContent (TIFF)	stDim:w stDim:h tiff:BitsPerSample tiff:ColorSpace tiff:ResolutionUnit tiff:SamplesPerPixel tiff:XResolution tiff:YResolution xap:Format xap:MetaDate xap:ModifyDate

TABLE 4.3 Metadata fields modified by AGS (Continued)

Command	Fields modified
convertSVGToPDF	pdf:Producer
convertRasterToPDF	
convertPSToPDF	
convertPDFToRaster	none
convertPSToRaster	
convertRasterToEPS	
convertSVGToRaster	
convertTo (SVG to Raster)	

Resolving Metadata Inconsistency

Some tools read and write only a particular format of metadata, ignoring any others. If such a tool has modified the metadata at any point, it can become inconsistent — that is, equivalent properties can have different values.

If a file contains inconsistent metadata, AGS resolves the inconsistency when you load the file, so that metadata in loaded content and metadata in files that you save from AGS is always consistent. If values of equivalent properties are inconsistent between formats, AGS applies precedence rules to determine the final value.

In general, if a file contains XMP metadata, AGS uses the XMP property value, unless it can determine that an equivalent property in another format has been updated more recently. The following table describes the rules that govern which value takes precedence.

TABLE 4.4 Metadata format precedence for file types

File format	Metadata precedence rules
PSD, TIFF	<p>AGS reads information in the following order, replacing earlier values with later values:</p> <ul style="list-style-type: none"> • kCaptionID image resource For older Photoshop files, the caption data is read from the image resource '1008' (kOldCaptionID) or '1020' (kPrintCaptionID). It cannot appear in both. • Image Description Tag (TIFF only) • IPTC-NAA Tag (TIFF only) TIFF information is read before image resource information, so an edit to the TIFF information is not recognized unless the image resource information is removed. <p>For Photoshop 6.5 XMP Data, the kCaptionID image resource is written out with a digest signature. The digest is regenerated. If it does not match, then the XMP information is dropped. Otherwise the XMP information is used instead of all previous data.</p>
SVG	AGS reads and preserves only XMP metadata.

TABLE 4.4 *Metadata format precedence for file types*

File format	Metadata precedence rules
PDF	<p>PDF files can have metadata stored in two places: the Info dictionary, and as XMP. AGS parses any existing XMP metadata first. If the Info dictionary information is older than the XMP data, AGS ignores it. If the Info dictionary information is newer, AGS uses it.</p> <ul style="list-style-type: none">• Keys that correspond with XMP keys go into the pdf namespace.• Keys that do not correspond with XMP go into the pdfx namespace.

Part II

Graphics

6

Changing Text in Images

This chapter describes how to set, replace, or modify text in Raster or SVG content. You can modify the text itself, using several different techniques, or use the `set` command to modify text display characteristics, such as the font and color.

This chapter contains the following sections:

Changing Text in Images	starting on page 113
Changing Text Attributes in Text Flows	starting on page 117

Changing Text in Images

Text is stored in text layers in PSD files, and in text flows (the `flowDef` element) in SVG files. AGS provides several ways to replace or modify text in a PSD or SVG file, automatically or explicitly:

Use the <code>replaceText</code> command to replace specific text in a specific text layer of a PSD file.	See “Replacing Text in Raster Content” on page 114 and the <i>Adobe Graphics Server Command Reference</i> .
Use automatic variable replacement with a text-type variable that you define in ImageReady or Illustrator to replace all text in an SVG text object or a PSD text layer.	See “Replacing Text Variables in PSD Files” on page 114 and “Replacing Text Variables in SVG Files” on page 116 .
Use automatic variable replacement with the AGS-defined variable <code>defaultTextVariable</code> to replace the topmost text layer in any PSD file.	See “Replacing Text Variables in PSD Files” on page 114 .
Use the <code>set</code> command to replace or modify the <code>textFlow</code> element in Raster content, or change the color of text by modifying the <code>colorOverlay</code> element.	See “Changing Text Attributes in Text Flows” on page 117 .
Use the <code>set</code> command to replace or modify the <code>flowDef</code> element in SVG content.	See “Changing Text Attributes in Text Flows” on page 117 .

For more information on these techniques and formats, see also the *Adobe Graphics Server Command Reference*, the chapter “Accessing Raster Content” in the *Adobe Graphics Server XML Grammars Reference*, and the SVG specification at the following URL:
<http://www.w3.org/TR/SVG/paths.html>

For text replacements, make sure the required fonts are available to the AGS server. If they are not, AGS uses a default font. For further information about fonts, see the *AGS Installation and Configuration Guide*.

Replacing Text in Raster Content

In Raster content, replaceable text is always in a text layer. (Bitmapped text in a pixel layer is not replaceable as text, although you can replace the image with another image containing different bitmapped text.) You can use the `replaceText` command to replace all of the text in a text layer, or to replace a specific string with another string.

EXAMPLE 6.1 Replacing all text in a layer

This example replaces all of the text in the layer named `Title` with the replacement text `The National Pastime`, and all of the text in the layer named `Description` with the replacement text `1997 National League Championship Series`.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="product_image.psd" />
  <replaceText target="/psd/layer[@name='Title']"
    text="The National Pastime" />
  <replaceText target="/psd/layer[@name='Description']"
    text="1997 National League Championship Series" />
  <saveContent name="imageWithNewTitle" appendExtension="true" />
</commands>
```

EXAMPLE 6.2 Replacing specific text in a layer

This example replaces the text string `69.99` in the layer named `Base Price` with the text `400`, and the text string `50` in the layer named `Percentage` with the text `40`.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="imageWithNewTitle.psd" />
  <replaceText target="/psd/layer[@name='Base Price']"
    text="69.99" textToReplace="400" />
  <replaceText target="/psd/layer[@name='Percentage']"
    text="50" textToReplace="40" />
  <saveContent name="replaceText1" appendExtension="true" />
</commands>
```

Replacing Text Variables in PSD Files

You can define a text-type variable in `ImageReady 7`, bind it to a text layer, and use the AGS automatic or explicit variable replacement facility to replace the variable value. Text variables can only be bound to the entire text layer, so you must replace all the text in the layer with this method. You can replace only the text, not any of the text attributes, such as the font or style.

For details of how to specify replacement values for variables and how to apply replacement values, see [Chapter 7, “Replacing Variables in SVG and PSD Files.”](#)

For any PSD file with one or more text layers, AGS automatically defines a text-type variable named `defaultTextVariable` and binds it to the topmost text layer (that is, the one with the highest index number). To replace the text in the topmost text layer automatically or explicitly, specify the new text value for the default variable named `defaultTextVariable` in the XML replacement data:

```
<data>
  <defaultTextVariable>new text</defaultTextVariable>
</data>
```

EXAMPLE 6.3 Replacing the automatic text variable value

The following example performs an explicit variable replacement, using XML replacement data that it loads inline in the `loadContent` command. It replaces the text in the topmost layer of the PSD file `imageWithCaption` with the string “Clearance!”

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent out="newTextData">
    <data>
      <defaultTextVariable>Clearance!</defaultTextVariable>
    </data>
  </loadContent>
  <loadContent source="imageWithCaption.psd" />
  <applyVariables data="newTextData" />
  <saveContent name="imageWithNewCaption" appendExtension="true" />
</commands>
```

Replacing Text in an SVG Text Flow

You can use the `set` command to replace the `flowDef` element or one of its subelements in SVG content. Depending on how you specify the target, you can replace just the text, or the text and its attributes, such as the font or style. See [“Changing Text Attributes in Text Flows” on page 117](#).

The following is an example of a text flow in an SVG file:

```
<switch id="text1" i:objectNS="http://ns.adobe.com/Flows/1.0/"
  i:objectType="pointText">
  <foreignObject requiredExtensions="http://ns.adobe.com/Flows/1.0/"
    x="0" y="0" width="1" height="1" overflow="visible">
    <flowDef xmlns="http://ns.adobe.com/Flows/1.0/">
      <region>
        <path d="M25.199,733.414"/>
      </region>
      <flow xmlns="http://ns.adobe.com/Flows/1.0/"
        font-family="'Myriad-Roman'"
        font-size="20.4375" text-align="right"
        text-align-last="right">
        <p><span>product name</span></p>
      </flow>
```

```

        </flowDef>
        <x:targetRef xlink:href="#XMLID_1_" />
    </foreignObject>
    <text id="XMLID_1_" transform="matrix(1 0 0 1 0 733.4141)">
    <tspan x="0" y="0" font-family="'Myriad-Roman'"
        font-size="20.4375">0.0</tspan></text>
</switch>

```

EXAMPLE 6.4 Replacing text in an SVG text flow

This example uses the `set` command to replace the text in the sample SVG text flow.

```

<?xml version="1.0" encoding="UTF-8"?>
<commands>
    <loadContent source="product_template.svg" />
    <set target="id('text1')/foreignObject/flowDef/flow/p/span/text()"
        value="My Product"/>
    <saveContent name="product_with_new_text" appendExtension="true" />
</commands>

```

EXAMPLE 6.5 Replacing an entire text flow in SVG content

This example uses the `set` command to replace the whole flow, both text and attributes, for the same SVG sample.

```

<?xml version="1.0" encoding="UTF-8"?>
<commands>
    <loadContent source="product_template.svg" />
    <set target="id('text1')/foreignObject/flowDef/flow">
        <flow xmlns="http://ns.adobe.com/Flows/1.0/" font-family="'Arial'"
            font-size="20.4375" text-align="right" text-align-last="right">
            <p><span>My Product</span></p>
        </flow>
    </set>
    <saveContent name="product_with_new_text" appendExtension="true" />
</commands>

```

Replacing Text Variables in SVG Files

You can define a text-type variable in Illustrator 10, bind it to a text flow, and use the AGS automatic or explicit variable replacement facility to replace the variable value. The replacement value can contain attribute values that affect format and style, as well as the text itself. A text variable can be replaced at four different levels of the text flow hierarchy:

- If the replacement data value is a `flow` element, AGS replaces the `flow` element in the source content with the new `flow` element.
- If the replacement data value is text only, AGS replaces the text of the first `span` child of the first `p` element (or the text of the first `p` element, if it has no `span`) of the `flow` element in the source content with the new text. It removes all other `span` and `p` elements.

- If the replacement data value is a `p` element, AGS replaces all the child elements of the `flow` element with the new `p` element.
- If the replacement data value is a `span` element, AGS replaces all the child elements of the first `p` element of the `flow` element with the new `span` element.

For details of how to specify replacement values for text variables and how to apply replacement values, see [Chapter 7, “Replacing Variables in SVG and PSD Files.”](#)

Changing Text Attributes in Text Flows

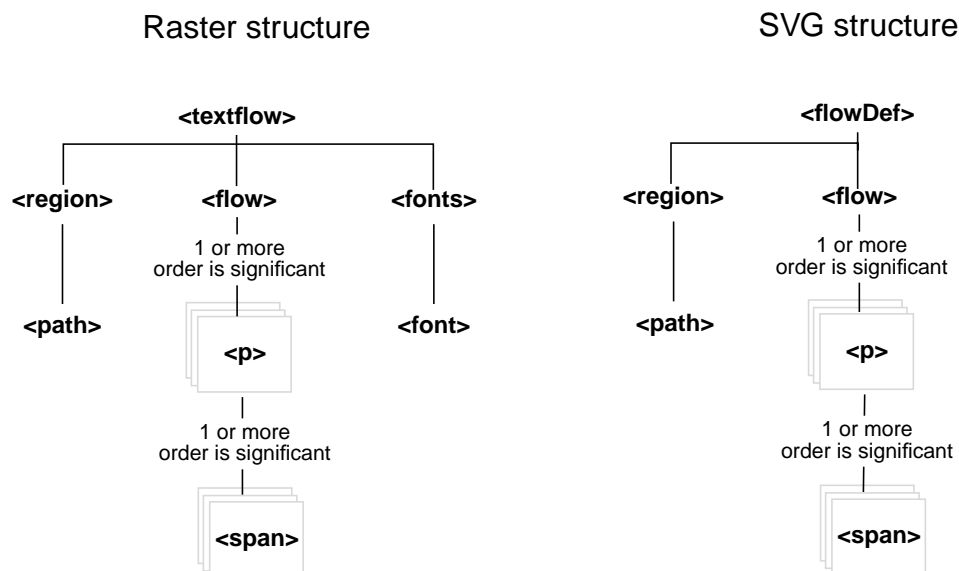
This section explains the XML structure of text flows in PSD and SVG files. Understanding this structure allows you to change the text in text flows in SVG and Raster content using the `set` command.

By setting elements and attributes of a text flow, you can affect the style and appearance of text, as well as the actual text. When you replace simple text bound to a text variable or use the `replaceText` command, you change only the text itself; the style is not affected.

The Structure of a Text Flow

A text flow is represented by a hierarchy of elements. A text flow hierarchy is defined by the SVG specification. For complete information, see the SVG specification at <http://www.w3.org/TR/SVG/paths.html>. AGS defines a similar hierarchy for Raster content, extended for PSD text information. [Figure 6.1](#) shows the hierarchical structure of the elements in a text flow, and specifies which elements apply to Raster and to SVG content.

FIGURE 6.1 XML structure of text flows in Raster and SVG content



As you can see from this figure, the element hierarchies for PSD and SVG text flows are very similar; they differ only in the name of the top element and the absence of the font elements in the SVG case. The `fonts` element can optionally provide a more reliable way of locating fonts for PSD files.

The actual text content of a text flow is specified in the `flow` element or its subelements. The other elements provide attributes, but no content.

The `region` and `path` elements define a position or shape for the text. AGS supports two types of text in a text flow, *area text* and *point text*:

- *Area text* — The area for area text is defined by a rectangle, drawn according to instructions in the `path` element. Area text wraps within the defined rectangle.

You create area text in Photoshop or Illustrator by clicking and dragging the cursor with the Text tool to create a rectangular area in a document and then typing text in that rectangle.

Within a text area, the `p` elements correspond to paragraphs. The number of `p` elements are determined by how many Return characters the text contains. When entering area text, pressing the Return key creates a new paragraph. If you press the Return key twice, you have three paragraphs, each of which is represented by its own `p` element.

- *Point text* — The area for point text is defined by a single origin point expressed in the `path` element. Point text contains a single line of text that does not wrap. You create point text in Photoshop or Illustrator by clicking in a document with the Text tool and typing the text. The text flow for point text has only one `p` element.

Each PSD text layer in a PSD file has one and only one text flow. If an image contains six text layers, there are six text flows in the image. SVG images do not have the notion of layers, but an SVG image can contain multiple text flows.

[Table 6.1](#) gives a brief summary of each of the elements in [Figure 6.1](#). For complete details of these elements and their attributes, see the chapter “Accessing Raster Content” in the *Adobe Graphics Server XML Grammars Reference*.

TABLE 6.1 Summary of elements and attributes in PSD and SVG text flows

Element & description	Attributes	PSD	SVG
<textflow> (PSD) or <flowDef> (SVG) Required The root of the text flow element hierarchy.	None		
<region> Required Contains one subelement that describes the position of point text, or the rectangular region of area text. Its attribute determines the direction of the text in the region (left-to-right or top-to-bottom), and region transformations (rotation, scaling, and coordinate translation).	writing-mode textMatrix	✓ 	✓ ✓
<path> Required Describes the origin point of point text or outline of area text.	d	✓	✓
<fonts> Optional A collection of all fonts used in the flow.	None	✓	
 Optional Identifies a single font that may be used in the text flow.	adobe-font-name adobe-font-script adobe-font-synthetic adobe-font-technology	✓ ✓ ✓ ✓	
<flow> Required Contains subelements that contain the actual text for the text flow. Attributes set at this level affect all text in the flow. Can contain text.	adobe-fractional-widths All attributes for p and span elements can also be specified on this element. Values specified in the p and span elements override those set at this level.	✓	✓

TABLE 6.1 Summary of elements and attributes in PSD and SVG text flows (Continued)

Element & description	Attributes	PSD	SVG
<p><p></p> <p>Required</p> <p>Defines a paragraph. Its attributes define the paragraph style.</p> <p>Contains one or more span subelements that contain the actual text. Each paragraph contains a style-span for each style of text in the paragraph.</p> <p>Can contain text.</p>	adobe-auto-leading-percent	✓	✓
	adobe-burasagari	✓	✓
	adobe-consecutive-hyphens	✓	✓
	adobe-every-line-composer	✓	✓
	adobe-hanging-roman	✓	✓
	adobe-hyphenate-capitalized	✓	✓
	adobe-hyphenate-word-size	✓	✓
	adobe-hyphenation-zone	✓	✓
	adobe-justification-glyph-scaling-desired	✓	
	adobe-justification-glyph-scaling-max	✓	
	adobe-justification-glyph-scaling-min	✓	
	adobe-justification-letter-spacing-desired	✓	✓
	adobe-justification-letter-spacing-max	✓	✓
	adobe-justification-letter-spacing-min	✓	✓
	adobe-justification-word-spacing-desired	✓	✓
	adobe-justification-word-spacing-max	✓	✓
	adobe-justification-word-spacing-min	✓	✓
	adobe-kinsoku-set	✓	
	adobe-leading-type	✓	✓
	adobe-mojikumi-set	✓	
<p><p></p> <p>(cont'd)</p>	adobe-preferred-kinsoku-order	✓	
	end-indent	✓	✓
	hyphenate	✓	✓
	hyphenation-push-character-count	✓	✓
	hyphenation-remain-character-count	✓	✓
	space-after	✓	✓
	space-before	✓	✓
	start-indent	✓	✓
	text-align	✓	✓
	text-align-last	✓	✓
	text-indent	✓	✓
	All span attributes can be specified on this element.		
	Values set in the span element override those set here.		

TABLE 6.1 Summary of elements and attributes in PSD and SVG text flows (Continued)

Element & description	Attributes	PSD	SVG
 Required The attributes define a character style that applies to the contained text. Can contain text.	adobe-baseline-direction	✓	
	adobe-faux-bold	✓	✓
	adobe-faux-italic	✓	✓
	adobe-font-baseline-option	✓	✓
	adobe-horizontal-scale	✓	✓
	adobe-ligatures	✓	
	adobe-no-break	✓	✓
	adobe-old-style-figures	✓	
	adobe-tsume	✓	
	adobe-vertical-scale	✓	✓
	adobe-y-underline	✓	
	baseline-shift	✓	✓
	fill	✓	✓
	font-family	✓	✓
	font-size	✓	✓
	font-variant	✓	
	kerning	✓	✓
	letter-spacing	✓	✓
 (cont'd)	line-height	✓	✓
	stroke		✓
	text-decoration	✓	✓
	text-transform	✓	

Setting Text Characteristics

You can use the `set` command to address individual attributes at any level of the text flow hierarchy, or to replace an entire element in the hierarchy, including both the attributes and text. For details of how to use the `set` command, see the *Adobe Graphics Server Command Reference*.

EXAMPLE 6.6 **Changing a font size**

This example loads a PSD file that contains a text layer named `caption`. It changes the font size in the first span of the text flow to 36 points.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="imageWithCaption.psd" />
  <set
    target="/psd/layer[@name='caption']/textFlow/flow/p/span[1]/@font-size"
    value="36.0" />
  <saveContent name="imageWithBiggerCaption" appendExtension="true" />
</commands>
```

EXAMPLE 6.7 **Changing the fill color for text**

This example sets the fill attribute to `#FF0000` (red) at the level of the `flow` element. Although fill is actually an attribute of `span`, when you set it at the higher level, the value applies to any span subelements that do not define a different value for it.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="imageWithCaption.psd" />
  <set target="/psd/layer[@name='caption']/textFlow/flow/@fill"
    value="#FF0000" />
  <saveContent name="imageWithBiggerCaption" appendExtension="true" />
</commands>
```

EXAMPLE 6.8 **Changing the overlay color in a text layer**

This example loads a PSD file that contains a text layer named `caption`, for which the creator defined a color overlay style in Photoshop. It changes the color overlay for the entire text layer to yellow.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="imageWithCaption.psd" />
  <set target="/psd/layer[@caption]/style/colorOverlay/@color"
    value="#FFFF00" />
  <saveContent name="imageWithYellowCaption" appendExtension="true" />
</commands>
```

For more information on color overlays, see [“Changing Colors in an Image” on page 98](#).

EXAMPLE 6.9 Replacing an entire text flow in Raster content

This example loads a source PSD file with a text layer named `caption`, and replaces the entire `textFlow` element in that layer. The new text flow provides the text “Now 30% off!,” changing the font size so that “30%” is bigger than the rest of the text.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="product_template.psd" />
  <set target="/psd/layer[@name='caption']/textFlow">
    <textFlow>
      <flow adobe-fractional-widths="false">
        <p text-align="center">
          <span font-family="'MyriadPro-Regular'"
            font-size="18.0">Now</span>
        </p>
        <p text-align="center">
          <span font-family="'MyriadPro-Regular'"
            font-size="30.0"> 30% </span>
        </p>
        <p text-align="center">
          <span font-family="'MyriadPro-Regular'"
            font-size="18.0">off!</span>
        </p>
      </flow>
    </textFlow>
  </set>
  <saveContent name="product_with_values" appendExtension="true" />
</commands>
```

EXAMPLE 6.10 Replacing text flow attributes in SVG content

This example uses the `set` command to replace both an entire flow and specific text flow elements for SVG content.

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <loadContent source="Envelope.svg" />
  <set target="id('Layer_1')/switch[3]/foreignObject/flowDef/flow">
    <flow xmlns="http://ns.adobe.com/Flows/1.0/"
      font-family="'Myriad-Roman'" font-size="12"
      line-height="11" text-align="right">
      <p><span>Mr. Smart</span></p>
      <p><span>100 Main Street</span></p>
      <p><span>City, ST 90000</span></p>
    </flow>
  </set>
  <set target="id('Layer_1')/switch[2]/foreignObject/flowDef
    /flow/p[2]/span[1]/text()"
    value="City, ST " />
  <set target="id('Layer_1')/switch[1]/foreignObject/flowDef
    /flow/p[1]/span/text()"
    value="A Company"/>
  <set target="id('Layer_1')/switch[1]/foreignObject/flowDef/flow/@fill"
    value="#FF0000" />
  <saveContent name="set_SVG1" appendExtension="true" />
</commands>
```

Notice that the target XPath includes the identifier. You could use an absolute path to the element:

```
target="/svg/g/switch[2]/foreignObject/flowDef/flow/p[2]/span[1]/text()"
```