Generic Device Application Programming Interface

Version 2.0.3
May 16, 2001

CreoScitex
3700 Gilmore Way
Burnaby, BC
Canada V5G 4M1

(604) 451-2700

Copyright 2001 Creo Products Inc. All rights reserved.

REVISION HISTORY FROM 2.0

Version	Date	Comments
2.0.1	June 2000	Changes to GDAPI_Connect.
2.0.1	August 2000	Removed the status code section; refer to function definitions in the documentation for lists of status codes. Added GDAPI_Initialized, GDAPI_SetImageTransferOptions, and GDAPI_GetImageTransferOptions.
2.0.2	September 2000	Added GDAPI_Authenticate
2.0.2	October 2000	Added the ECO document control number.
2.0.3	March 2001	Added GDAPI_GetNumDefaultAttributeValues, GDAPI_GetDefaultAttributeValue and GDAPI_DeleteCurrentAttributeValue.

CONTENTS

REVISION HISTORY FROM 2.0 2

1. SCOPE 7
   1.1 Advantages of the GDAPI 7

2. HIGH LEVEL DESIGN 8
   2.1 Device Specific Implementation (DSI) 8
   2.2 Raster Source 8
   2.3 Print Session 8
   2.4 GDAPI Services 8
   2.5 Local GDAPI Configuration 9
   2.6 Client Server GDAPI Configuration 10
   2.7 Raster Source Requirements 10
   2.8 DSI Requirements 11
      2.8.1 Synchronous API Requirement 12
   2.9 GDAPI Requirements 12
      2.9.1 Unicode 12
      2.9.2 C Interface 12

3. GDAPI 13
   3.1 Data Types & Constants 13
   3.2 Status Codes 13
      3.2.1 DSI Status Codes & String IDs 14
   3.3 Internationalization 15
      3.3.1 Locale Identifier 15
      3.3.2 Locale Sensitive Strings 17
   3.4 Print Sessions 17
      3.4.1 Print Session Control 17
   3.5 Print Session Data 19
      3.5.1 Attribute Settings for Print Sessions 23
      3.5.2 Resolution Data 23
      3.5.3 Image Dimensions 23
      3.5.4 Image Offset 24
      3.5.5 Image Clipping 26

3.5.6 Image Data 27
3.6 Device State & Sub-state 29
3.6.1 Device State Overview 29
3.6.2 Device Sub-State 29
3.6.3 Device States 31
3.7 Device Attributes Overview 31
3.7.1 Attribute Function Overview 32

4. SCENARIOS 34
4.1 Startup 34
4.1.1 Raster Source Initialization 34
4.1.2 Device Connection 36
4.2 Device Disconnect & Raster Source Disconnect 37
4.3 Print Session 38
4.4 Device Shutdown 39
4.5 Cancel Image 40
4.5.1 Cancel Image: Initiated by the Raster Source 40
4.5.2 Cancel Image: Initiated by the Imaging Device 41
4.6 Cancel Print Session 42
4.6.1 Cancel Print Session: Initiated by the raster source 42
4.6.2 Cancel Print Session: Initiated by the Imaging device 43

5. GDAPI FUNCTIONS 44
5.1 GDAPI_GetVersion 44
5.2 GDAPI_Initialized 45
5.3 GDAPI_SetVersion 45
5.4 GDAPI_Authenticate 46
5.5 GDAPI_Connect 47
5.6 GDAPI_Disconnect 48
5.7 GDAPI_GetNumAvailableDevices 48
5.8 GDAPI_GetAvailableDevices 49
5.9 GDAPI_RegisterDeviceStateCallback 50
5.10 DeviceStateCallback 51
5.11 GDAPI_GetDeviceState 52
5.12 GDAPI_RegisterPrintSessionCallback 53
5.13 PrintSessionCallback 53
5.14 GDAPI_RegisterAttributeChangedCallback 55
5.15 AttributeChangedCallback 55
5.16 GDAPI_GetNumAvailableAttributeValues 56
5.17 GDAPI_GetAvailableAttributeValue 57
5.18 GDAPI_GetNumDefaultAttributeValues 59
5.19 GDAPI_GetDefaultAttributeValue 60
5.20 GDAPI_GetNumCurrentAttributeValues 61
5.21 GDAPI_GetCurrentAttributeValue 63
5.22 GDAPI_SetCurrentAttributeValue 64

5.23 GDAPI_DeleteCurrentAttributeValue 66
5.24 GDAPI_DeviceConnect 68
5.25 GDAPI_DeviceDisconnect 69
5.26 GDAPI_QueuePrintSession 69
5.27 GDAPI_UpdatePrintSessionData 71
5.28 GDAPI_CancelPrintSession 72
5.29 GDAPI_ReadyForRelease 73
5.30 GDAPI_StartPrintSession 74
5.31 GDAPI_StartImageTransfer 75
5.32 GDAPI_SetImageTransferOptions 75
5.33 GDAPI_GetImageTransferOptions 78
5.34 GDAPI_SendImageData 79
5.35 GDAPI_EndImageTransfer 81
5.36 GDAPI_EndPrintSession 81
5.37 GDAPI_CancelCurrentImage 82
5.38 GDAPI_GetLocalizedStringLength 83
5.39 GDAPI_GetLocalizedString 84
5.40 GDAPI_SendMessage 85
5.41 GDAPI_GetMessageLength 86
5.42 GDAPI_GetMessage 87

6. ERROR HANDLING 89

7. ATTRIBUTES 90
7.1 Core Attribute Summary 90
7.2 Non-Core Attribute Summary 90
7.3 Data Scope 91
7.3.1 Overview 91
7.3.2 Examples 92
7.3.3 Data Scope Errors 92
7.3.4 Summary 93
7.4 Default Values 95
7.5 Core Attribute Details 95
7.5.1 DEVICE_ATTRIBUTE 95
7.5.2 QUEUE_LENGTH 98
7.5.3 RESOLUTION 99
7.5.4 Relationship Between Media Attributes 100
7.5.5 DEVICE_MEDIA 100
7.5.6 Implications of “Negative Media” 104
7.5.7 MEDIA_SIZE 105
7.5.8 MEDIA_THICKNESS 108
7.5.9 BIT_DEPTH 109
7.5.10 INTERLEAVING 109
7.6 Non-Core Attribute Details 111
7.6.1 MEDIA_AVAILABLE 111

7.6.2 COLOR_NAME 112
7.6.3 INVERT_RASTER 112
7.7 Standard Attribute Type Overview 114
7.7.1 Reading Standard Attributes 114
7.7.2 Writing Standard Attributes 114
7.7.3 Example: List Attribute 114
7.7.4 Example: Non-List Attribute 115
7.8 Standard Attribute Types 116
7.8.1 AttributeType_String 116
7.8.2 AttributeType_StringList 117
7.8.3 AttributeType_Long 117
7.8.4 AttributeType_LongList 117
7.8.5 AttributeType_Double 118
7.8.6 AttributeType_DoubleList 118
7.8.7 AttributeType_Bool 118
7.8.8 AttributeType_RangeLong 118
7.8.9 AttributeType_RangeDouble 119

8. GLOSSARY 120

1. SCOPE

This document describes the basic functionality of the Generic Device Application Programming Interface (**GDAPI**). The intention of a GDAPI is to create a general interface framework allowing an arbitrary raster source to easily connect to and image raster data on a variety of output devices.

This document describes version 2.0.3 of the GDAPI.

1.1 Advantages of the GDAPI

A direct connection from workflow software to a CreoScitex imaging device will provide a customer with an increase in productivity and more choices in the way that a job is processed. These advantages are a result of the following characteristics of a GDAPI connection.

- **Speed**
  RIPs using the GDAPI send raster data directly to the device rather than generating intermediate TIFF files that are then processed via a Hot Folder.

- **Flexibility**
  The GDAPI allows a workflow to set job options on a per-job basis and a per-separation basis within a given job. This is in contrast to a Hot Folder approach that forces options on a per Hot Folder basis.

- **Simplified Configuration**
  A TIFF connection requires configuration of Hot Folders and 'Print Setups'. Print Setups are used by each Hot Folder to associate a Media Size and TIFF processing options with the Hot Folder. A direct connection eliminates the need for Hot Folders and allows the user to select the Media Size and job processing options directly from the workflow.

- **Centralized Control**
  The control of a job (configuring, queuing, processing and archiving) is centralized in the GDAPI client, rather than being shared between the workflow and Print Console. This simplifies configuration and maintenance and minimizes the chance for unwanted interactions between workflow settings and Print Console settings.

2. HIGH LEVEL DESIGN

This section provides a high level overview of the users of the GDAPI, services provided by the GDAPI, and some GDAPI configurations.

2.1 Device Specific Implementation (DSI)

A **DSI** (Device Specific Implementation) is a GDAPI implementation which knows how to control one or more devices.

A DSI shall be implemented as a DLL – this common implementation model will allow a raster source to easily deal with different DSIs.

2.2 Raster Source

A raster source is a component which can produce raster and which uses the GDAPI to:

- Connect to an output device.
- Determine output device capabilities.
- Send raster to an output device.

2.3 Print Session

A raster source can use the GDAPI to queue one or more print sessions for a given imaging device. A print session consists of one or more images – also called separations.

2.4 GDAPI Services

At a high level, the GDAPI provides the following services:

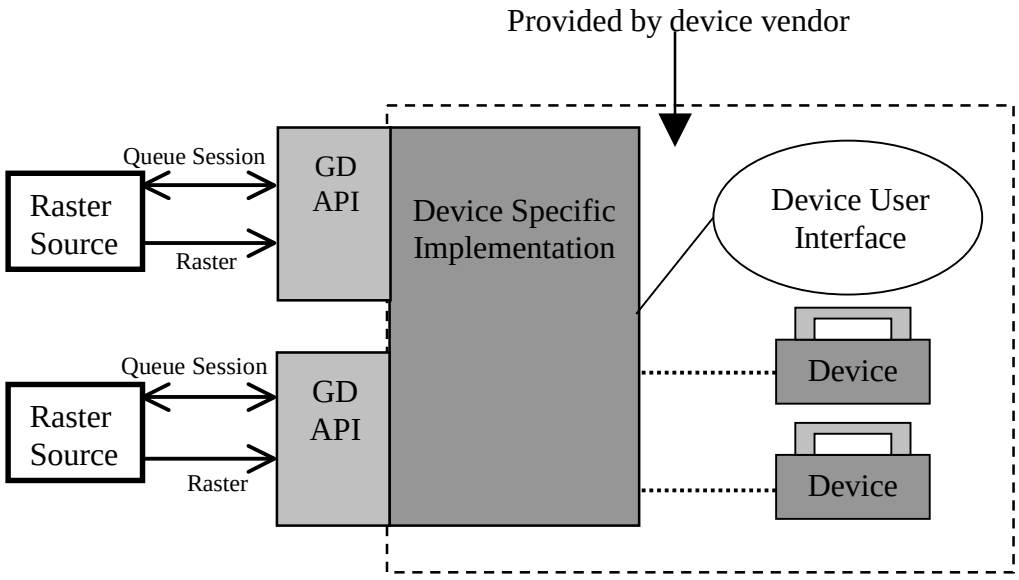
- Basic functions needed to transfer raster data from a raster source to the DSI – and subsequently to the output device.
- The ability for a raster source to queue one or more print sessions to an output device; the number of sessions which can be queued depends upon the capabilities of the DSI and device. A print session consists of a number of images or separations. The raster source queues a session to the DSI and then the DSI requests that the raster source transfer the raster for a particular session (note that the DSI may request print sessions in an order which is different from the queued order). The raster source then transfers the raster images for the session, using the appropriate GDAPI functions.

The Device Specific Implementation (DSI) may optionally implement a user interface (UI) to provide some of the following functionality to the operator:

- provide an accurate graphical representation of the device's state,
- perform device specific configuration (media configuration and specific imaging configuration),
- perform device diagnostics, and
- control the output device's queue, if required.

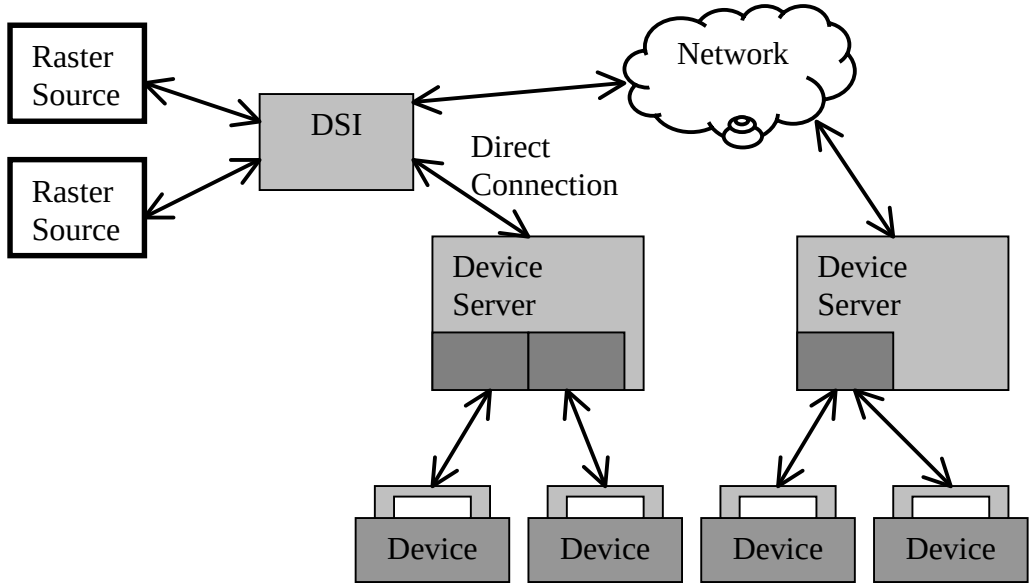
2.5 Local GDAPI Configuration

The following diagram depicts a local configuration (the raster source and DSI are one the same computer) of two raster sources using a DSI to communicate with some imaging devices.



2.6 Client Server GDAPI Configuration

The GDAPI can also support a client server model as depicted by the following diagram.



The DSI:

- Uses a network to communicate with one or more Device Servers; The DSI may also be running on a Device Server – in which case a direct connection to the Device Server is possible.
- Allows the raster source to determine which devices are accessible from each Device Server.
- Accepts function calls from a raster source.

A Device Server:

- Runs device control software for one or more Devices.

2.7 Raster Source Requirements

Each raster source must run in a separate process; this is a requirement, as the GDAPI does not provide a way to distinguish between raster sources that are running in the same process. A DSI, however, will need to distinguish between raster sources, see “DSI Requirements” for details.

If the raster source encounters a fatal error it must call GDAPI_Disconnect as part of its shutdown. This will provide the DSI with the notification it requires to discard resources and cancel any print sessions associated with the raster source.

A raster source must implement the following callbacks, other callbacks are optional:

- PrintSessionCallback
- AttributeChangedCallback

The DSI can release a print sessions in an order which is different from the queued order; If a raster source cannot cope with a change in the order of queued print sessions then it can avoid any change in order by queuing print sessions one at a time. The next print session can be queued after the first session has been released.

2.8 DSI Requirements

A DSI should translate all of the GDAPI function calls into specific instructions to control the output device.

A DSI must support the registration and use of all callback functions defined in this API. Any non-callback functions not supported must at least be implemented as a "stub" and return the status code: GDAPI_UNSUPPORTED_FUNCTION.

A DSI should attempt to detect when a raster source has exited. This will be useful in cases where the raster source has encountered a fatal error and was not able to signal its exit. When the DSI detects such a case it will delete any information associated with the raster source, including queued sessions. If the raster source was the owner of the current print session, the DSI will abort the current session and process print sessions queued by other raster sources – if any.

A DSI must document its implementation of the GDAPI:

- Which non-core attributes it supports (see the section on "Device Attributes").
- How the GDAPI is supported. Ideally a DSI should match the GDAPI specification exactly; any variation should be clearly stated – to allow a raster source to adjust to the differences.

The DSI shall be implemented as a DLL; some possible models are:

- The DLL contains the device control code.
- The DLL has inter-process communications (IPC) to another process which implements the actual device control; the device control component may or may not have a GUI.

2.8.1 Synchronous API Requirement

It is a requirement that the DSI implement all GDAPI functions as synchronous functions.

This means that the task performed by the function must be completed before the function returns, or must appear to the caller of the function to have completed. For example, an implementation of the GDAPI_SendImageData function may copy the image data and return; the imaging data may be sent to the device later, but the caller can consider that the data is sent and that the buffer passed to GDAPI_SendImageData can be re-used or deleted.

2.9 GDAPI Requirements

2.9.1 Unicode

All strings in the GDAPI are Unicode strings.

2.9.2 C Interface

The GDAPI uses ‘C’ type function names, I.E. no name mangling.

3. GDAPI

This section covers requirements and concepts that are not specific to one given GDAPI function.

3.1 Data Types & Constants

Here are the constants that must be defined for a DSI:

```
// Used for arguments or fields when it is appropriate that
// no value be specified.
//
#define GDAPI_NO_VALUE –1

#define GDAPI_VARIABLE_THICKNESS –2

// Constants for string lengths.
//
#define GDAPI_SHORT_STRING  256

// The Facility code for a GDAPI_Status code which is used for
// a Device State Detail code.
//
#define GDAPI_SUB_STATE_FACILITY 0x121

// Names of attributes.
//
const wchar_t ATTR_DEVICE_ATTRIBUTE[] = L”DEVICE_ATTRIBUTE”;
const wchar_t ATTR_QUEUE_LENGTH[]     = L”QUEUE_LENGTH”;
const wchar_t ATTR_RESOLUTION[]       = L”RESOLUTION”;
And other core attribute names as defined in this specficiation.

// Standard process color names.
//
const wchar_t COLOR_C[] = L”CYAN”;
const wchar_t COLOR_M[] = L”MAGENTA”;
const wchar_t COLOR_Y[] = L”YELLOW”;
const wchar_t COLOR_K[] = L”BLACK”;
```

3.2 Status Codes

GDAPI defines function status codes and string Ids with the same bit level format. This means a DSI need support only one mechanism for localization; I.E a status code or string Id can both be translated into a locale specific string using the same mechanism.

GDAPI_Status: is a 32 bit signed value defined as:

```
typedef long GDAPI_Status;
```

GDAPI_Status is based on the Windows HRESULT status and has the following characteristics:

- Allows the definition of a custom facility Code.
- The severity code is 2 bits rather than 1.
- HRESULT status codes can be mapped into a GDAPI_Status.

The following table shows the bit level layout of a GDAPI_Status code.

Section	Bits	Purpose
Code	16 bits: 0-15	The error code for a given Facility. Core status codes start at Code 0, DSI specific status codes start at GDAPI_DSI_DEFINED_STATUS_CODE
Facility	12 bits: 16-27	**The Facility code, which must be unique for each DSI.** Apply to GDAPI@CreoScitex.com for a DSI specific range of Facility codes.
Reserved	1 bit: 28	For internal use.
Customer Code Flag	1 bit: 29	
Severity	2 bits: 30-31	The severity, as defined below.

Severity codes are defined as:

```
#define GDAPI_SEVERITY_SUCCESS       0
#define GDAPI_SEVERITY_INFORMATIONAL 1
#define GDAPI_SEVERITY_WARNING       2
#define GDAPI_SEVERITY_ERROR         3
```

There are core status codes (see the header files provided with the GDAPI SDK) and a DSI may define its own status codes – with the restrictions outlined in the table above.

The GDAPI header files contain macros for creating a GDAPI_Status value, extracting sections from a GDAPI_Status value, checking for the success or failure indicated by the severity of a status value, and converting from an HRESULT to a GDAPI_Status.

3.2.1 DSI Status Codes & String IDs

Please refer to the GDAPI header files provided with the GDAPI SDK for a complete list of core status codes. Status codes that should be returned for a given function are listed with the function description in this document.

If a DSI defines status codes or string Ids that do not exist in the “core” GDAPI_Ids.h file, the DSI must:

- Apply to GDAPI@CreoScitex.com for a range of values for the Facility field; this will ensure that codes are unique across all implementations.
- Start with a value of GDAPI_DSI_DEFINED_STATUS_CODE in the Code field.

3.3 Internationalization

Note that GDAPI does not provide localization support for the display of the following; support for these must be provided by operating system facilities:

- Numerical values.
- Date values.

3.3.1 Locale Identifier

A locale identifier is required to support localization and for GDAPI functions which return locale specific versions of strings.

The GDAPI uses the Microsoft Windows approach for locale identification, but does not require any Windows headers to support this approach; this allows for a DSI on a non-Windows platform.

A locale identifier (LCID) is a “double word” value containing the language identifier in the lower word and the sorting ID as well as a reserved value in the upper word. The identifier supplied in an LCID is a standard international numeric abbreviation. Use the MAKELCID macro to create LCID values from a language identifier and a sorting ID.

A DSI will provide the following support for building a locale ID (*note that this is provided by winnt.h when developing on the Windows platform*):

```
typedef unsigned long  GDAPI_LocaleId;

typedef unsigned long  DWORD;  // double word
typedef unsigned short WORD;   // single word

//
//  A language ID is a 16 bit value which is the combination of a
//  primary language ID and a secondary language ID.  The bits are
//  allocated as follows:
//
//       +-----------------------+-------------------------+
//       |     Sublanguage ID    |   Primary Language ID   |
//       +-----------------------+-------------------------+
//        15                   10 9                       0   bit
//
//  Language ID creation/extraction macros:
//
//    MAKELANGID
//       construct language id from a primary language id and
//       a sublanguage id.
//
//    PRIMARYLANGID
//       extract primary language id from a language id.
//
//    SUBLANGID
//       extract sublanguage id from a language id.
//

#define MAKELANGID(p, s) \
   ((((WORD  )(s)) << 10) | (WORD  )(p))
```

```
#define PRIMARYLANGID(lgid) ((WORD  )(lgid) & 0x3ff)

#define SUBLANGID(lgid) ((WORD  )(lgid) >> 10)

//
//  A locale ID is a 32 bit value which is the combination of a
//  language ID, a sort ID, and a reserved area.  The bits are
//  allocated as follows:
//
//       +-------------+---------+-------------------------+
//       |   Reserved  | Sort ID |      Language ID        |
//       +-------------+---------+-------------------------+
//        31         20 19     16 15                       0   bit
//
//
//  Locale ID creation/extraction macros:
//
//    MAKELCID
//       construct the locale id from a language id and a sort id.
//
//    MAKESORTLCID
//       construct the locale id from a language id, sort id, and
//       sort version.
//
//    LANGIDFROMLCID
//       extract the language id from a locale id.
//
//    SORTIDFROMLCID
//       extract the sort id from a locale id.
//
//    SORTVERSIONFROMLCID
//       extract the sort version from a locale id.
//
#define MAKELCID(lgid, srtid) \
   ((DWORD)((((DWORD)((WORD  )(srtid))) << 16) |  \
   ((DWORD)((WORD  )(lgid)))))

#define MAKESORTLCID(lgid, srtid, ver) \
   ((DWORD)((MAKELCID(lgid, srtid)) |  \
   (((DWORD)((WORD  )(ver))) << 20)))

#define LANGIDFROMLCID(lcid) ((WORD  )(lcid))

#define SORTIDFROMLCID(lcid) \
   ((WORD  )((((DWORD)(lcid)) >> 16) & 0xf))

#define SORTVERSIONFROMLCID(lcid) \
   ((WORD  )((((DWORD)(lcid)) >> 20) & 0xf))
```

A DSI will also provide, for each locale and sorting option supported, copies of the Windows constants for:

- Primary language IDs.
- Sub-language IDs.
- Sort IDs.

Note that the Sort version ID is reserved and must be 0.

3.3.2 Locale Sensitive Strings

A raster source may want to display locale specific versions of strings for values provided by the GDAPI; such values may include:

- Enumerated values.
  Each value in an enumeration is associated with a string ID.
- Attribute names (introduced later).
- String values.

To support localization a DSI must store all locale sensitive strings in a string table:

- Each string will have a GDAPI_StringId as a key. **The format of this key is the format of a function status code as described in the "**Error! Reference source not found.**" section.**
- Each string will be stored as a wchar_t string.

The GDAPI_GetLocalizedString function returns a localized string when supplied with a locale ID and a string ID.

3.4 Print Sessions

This section provides some overview material for print sessions and also discusses the print session data which is used by the GDAPI_QueuePrintSession and GDAPI_UpdatePrintSessionData function calls.

3.4.1 Print Session Control

This section provides an overview of "print session control" in the context of a print session; further details are provided in the documentation of each function.

Raster Source	**DSI**
A print session is queued by GDAPI_QueuePrintSession. The raster source supplies some details of the print session and **a "print session ID" is returned, say 1.**	
The raster source can cancel the print session at any time after it has been queued by calling GDAPI_CancelPrintSession and supplying the print session ID.	PrintSessionCallback **can** issue the CANCEL_PRINT_SESSION command at any point after the print session has been queued.
The raster source can use GDAPI_SetCurrentAttributeValue to set attribute values which apply to this print	

session or separations within this print session. Each attribute can have a different scope – attributes with a scope of PrintSession can be set until the raster source calls GDAPI_ReadyForRelease, attributes with a scope of Separation_1 can be set until the **first** call to GDAPI_StartImageTransfer, attributes with a scope of Separation_n can be set until the **nth** call to GDAPI_StartImageTransfer.	
When the raster source has finished setting attribute values that have a scope of PrintSession, it calls GDAPI_ReadyForRelease.	
The raster source does not do anything else with this print session until signaled by the DSI via the PrintSessionCallback.	
The raster source can queue another print session while waiting for the "release" of other queued print sessions.	
	PrintSessionCallback issues the RELEASE command for print session 1.
The raster source calls GDAPI_StartPrintSession and starts generating raster.	
When the raster source is ready to send raster for the first image or separation it calls GDAPI_StartImageTransfer.	
	When the DSI is ready to receive raster it calls PrintSessionCallback and issues the SEND_RASTER command. This command is send for each image or separation in the print session.
The raster source calls GDAPI_SendImageData to send the data for this separation; this function may be called several times for a given image.	
The raster source can call GDAPI_CancelCurrentImage to cancel the current image if a problem occurs while sending raster data.	PrintSessionCallback **can** issue the CANCEL_CURRENT_IMAGE command if a problem occurs while receiving raster data.

The raster source calls GDAPI_EndImageTransfer to signal the end of a separation.	
More images are transferred, as above, or the raster source calls GDAPI_EndPrintSession to finish the current print session.	

3.5 Print Session Data

Print Session data is used by:

- GDAPI_QueuePrintSession
  to queue a print session.
- GDAPI_UpdatePrintSessionData
  to update the data for a previously queued print session. This is useful as a print session can be queued before all data is known. This allows an output device to prepare for a print session without having the raster source process all of the data in an input file.

Here is the data structure used:

```
typedef struct
{
    wchar_t               PrintSessionName[GDAPI_SHORT_STRING];
    unsigned short        FieldSetFlag;

    wchar_t               MediaSizeName[GDAPI_SHORT_STRING];
    GDAPI_MediaThickness  MediaThickness;
    short                 SignatureId;
    short                 SheetId;
    GDAPI_Resolution      Resolution;
    GDAPI_ImageDimension  ImageDimensions;
    GDAPI_ImageOffset     ImageOffset;
    GDAPI_ImageClipping   ImageClipping;
    short                 BitDepth;
    GDAPI_Interleaving    Interleaving;

    short                 NumImages;
    GDAPI_Image*          Image;

} GDAPI_PrintSessionData;
```

PrintSessionName

The name of the print session. The primary use of this string is to identify the session in any user interface. This could be the customers job name.

FieldSetFlag

Some fields in this structure are optional, or the raster source may not know their value when the print session is queued and want to specify the values later by using GDAPI_UpdatePrintSessionData. This field is used to indicate which of the following fields contain valid data. FieldSetFlag is set to FIELD_ALL_FIELDS if all fields contain data.

GDAPI_UpdatePrintSessionData can be used to change some field settings or can be used to provide information on fields that where not specified when the print session was queued. Refer to the table at the end of this section for details of when data can be provided or changed.

MediaSizeName

The media to image on. MediaSizeName must be one that is returned for the MEDIA_SIZE attribute. Note that the media does not need to be currently available on the device when the print session is queued – if the media cannot be made available when the device requests the media, then a DSI can cancel the print session. Set the FIELD_MEDIA_SIZE bit in FieldSetFlag if this field contains valid data.

MediaThickness

This field is required only when the MEDIA_SIZE attribute, referred to by the MediaSizeName field, contains a thickness value of GDAPI_VARIABLE_THICKNESS. In this case the thickness must be specified with this field – the MediaThickness field. Set the FIELD_MEDIA_THICKNESS bit in FieldSetFlag if this field contains valid data.

Valid thickness values are obtained by reading the MEDIA_THICKNESS attribute. Note that the MEDIA_SIZE selected is a specific type of DEVICE_MEDIA and that the thickness values are defined for different types of DEVICE_MEDIA; you will need to select a thickness value from the list of thickness values defined for the appropriate DEVICE_MEDIA.

SignatureID

ID of the signature; specify GDAPI_NO_VALUE if it is unknown.
Set the FIELD_SIGNATURE_ID bit in FieldSetFlag if this field contains valid data.

SheetID

ID of the sheet in the current signature; specify GDAPI_NO_VALUE if it is unknown. Set the FIELD_SHEEET_ID bit in FieldSetFlag if this field contains valid data.

Resolution

The resolution of all output images in this print session; a different resolution can be specified for both scan directions: fast scan and slow scan. Set the FIELD_RESOLUTION bit in FieldSetFlag if this field contains valid data. Valid resolution options are obtained by getting the attribute values for the RESOLUTION attribute.

ImageDimensions

The dimensions of each image in this print session. Set the FIELD_IMAGE_DIMENSION bit in FieldSetFlag if this field contains valid data.

ImageOffset

This is an optional field which can specify an amount to offset the image from the default position. See the section titled “Image Offset” for details. Set the FIELD_IMAGE_OFFSET bit in FieldSetFlag if this field contains valid data.

ImageClipping

This is an optional field which can specify an amount to clip from each image in a print session. This structure provides 4 fields to specify a different clipping amount for each edge of the image; see the section titled “Image Clipping” for details. Set the FIELD_IMAGE_CLIPPING bit in FieldSetFlag if this field contains valid data.

BitDepth

Bit depth is 1 for half tone images and greater than 1 for con-tone images – con-tone images are sent to some types of proofing devices where each pixel of raster data contains several bits of color information. Set the FIELD_BIT_DEPTH bit in FieldSetFlag if this field contains valid data.

Interleaving

Interleaving indicates how the **RIP** has combined the raster data to form the final images in a print session. Interleaving options supported by a DSI are obtained from the INTERLEAVING attribute. This information can be used by the DSI to rework the image data, if a given device does not support the interleaving method specified. Set the FIELD_INTERLEAVING bit in FieldSetFlag if this field contains valid data.

NumImages

The number of images in this print session. Set the FIELD_IMAGES bit in FieldSetFlag if the NumImages and Image fields contain valid data.

Image

An array of GDAPI_Image structures; the array size must match the value provided in the NumImages field. Set the FIELD_IMAGES bit in FieldSetFlag if the NumImages and Image fields contain valid data.

Parts of the print session data may be displayed by the DSI, if it has a GUI, and it may be printed on the media by the imaging device.

The following table shows the cut-off points for each type of print session data. The cut-off points show in the table are:

- **Queue Print Session** – data with this cut off point is specified in GDAPI_QueuePrintSession and can not be changed afterward.
- **Ready For Release** – data with this cut off point can be specified by GDAPI_QueuePrintSession and can be changed via GDAPI_UpdatePrintSessionData until GDAPI_ReadyForRelease is called.

- **Separation 1** – data with this cut off point can be changed via GDAPI_UpdatePrintSessionData until the **first** call to GDAPI_StartImageTransfer.
- **Separation n** – data with this cut off point can be changed for separation 'n' via GDAPI_UpdatePrintSessionData until the **nth** call to GDAPI_StartImageTransfer.

The MediaThickness must be specified only if the media varies in thickness, I.E. the MEDIA_SIZE attribute has a thickness of GDAPI_VARIABLE_THICKNESS.

Data	**Cut Off**			
	Queue Print Session	**Ready For Release**	**Separation 1**	**Separation n**
PrintSessionName	✔			
MediaSizeName		✔		
MediaThickness		✔		
SignatureID			✔	
SheetID			✔	
Resolution			✔	
ImageDimensions			✔	
ImageOffset			✔	
ImageClipping			✔	
BitDepth			✔	
Interleaving			✔	
NumImages & Image Data				✔

3.5.1 Attribute Settings for Print Sessions

Please note that some attributes can also be set to control some aspects of a print session or change device settings for the duration of a print session.. The attributes which are available for print session control are device dependent. Please see the section “Attributes” for details of attributes.

3.5.2 Resolution Data

Resolution data is specified as part of the print session data.

```
typedef struct
{
    double                FastScan;
    double                SlowScan;
    GDAPI_ResolutionUnits Units;

} GDAPI_Resolution;
```

Related Data:

```
typedef enum
{
    ResolutionUnits_DotsPerInch         = IDS_RESOLUTION_DPI,
    ResolutionUnits_DotsPerCentimeter   = IDS_RESOLUTION_DPCM,
    ResolutionUnits_SpotSizeMicrometers = IDS_RESOLUTION_UNITS_SSM

} GDAPI_ResolutionUnits;
```

- **FastScan**
  The resolution around the drum.
- **SlowScan**
  The resolution across the drum.
- **Units**
  This is the units of measure for the resolution.

  Note that “Spot Size Micrometers” is the resolution expressed as the size of the spot in micrometers, I.E. a spot size of 10 micrometers would allow 100 spots / millimeter or 100 spots / centimeter.

3.5.3 Image Dimensions

All images in a print session have the same size which is specified by the ImageDimensions field:

```
typedef struct
{
```

```
        unsigned long   SlowScanPixel;
        unsigned long   FastScanPixel;

    } GDAPI_ImageDimension;
```

The image dimensions specify the number of pixels in the slow scan and fast scan directions.

If the image is larger than the image area, then it is up to a DSI how this is handled:

- Return an error indicating that the image is too large: GDAPI_IMAGE_TOO_LARGE.
- Clip the image which falls outside the image area; the edges which are clipped will depend upon where the image is placed on the medium, placement is controlled by the print session data ImageOffset field.

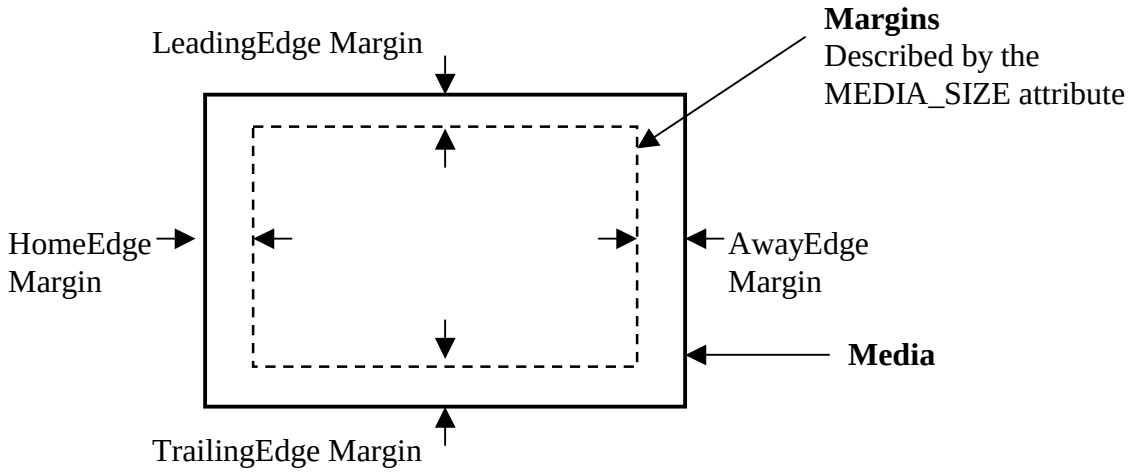
3.5.4 Image Offset

The ImageOffset field is used to specify an amount to offset each image from its default location on the media.

If this value is **not** set for a print session then the image will be placed in the default position as defined below.

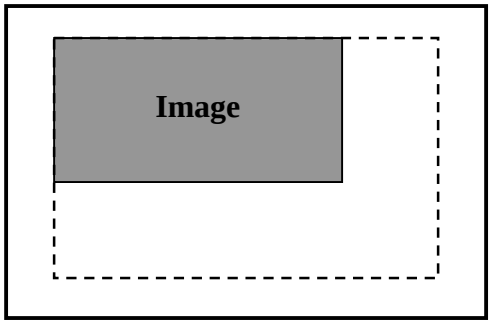
The ImageOffset field is used to control the amount an image is offset from the default position. The default position is defined by the margins described by the MEDIA_SIZE attribute and is detailed below.

Offsets are relative to the margins specified by the MEDIA_SIZE attribute. The margins are defined by the device and a given margin can be 0. The size of the

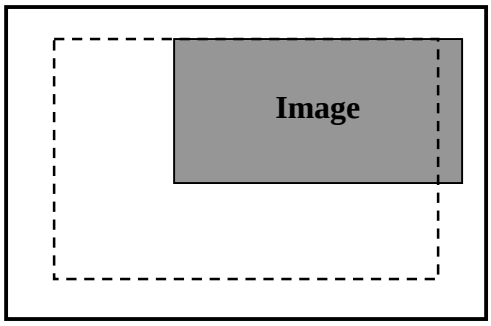


margins is typically dictated by 1) clamping requirements and 2) media requirements (I.E. to avoid imaging close to the edge of the media).

The next diagram shows how an image is placed on the plate if an image offset is not specified or if both the HomeEdge and LeadingEdge offsets are zero; this is the **default position**.



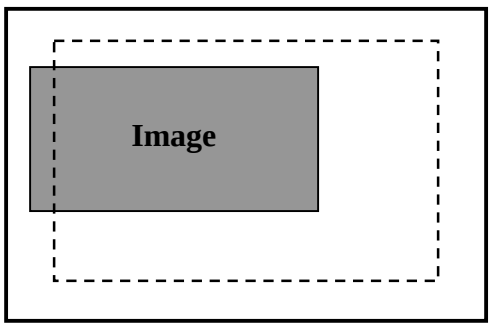
The next diagram shows how an image is placed on the plate if the HomeEdge offset is positive and the LeadingEdge offset is zero.



There are a few things to note about this offset:

- Whether the part of the image which falls outside the **margin** is drawn or not is device dependent.
- The area on the medium to the left of the image will not be treated any differently than other areas on the medium which fall outside of the Image; this has implications for some types of media – the image size must be increased to correctly image the medium.

The next diagram shows how an image is placed on the plate if the HomeEdge offset is negative and the LeadingEdge offset is positive .



Data:

```
typedef struct
{
    double               LeadingEdge;
    double               HomeEdge;
    GDAPI_UnitOfMeasure  UnitOfMeasure;

} GDAPI_ImageOffset;
```

- LeadingEdge
  The offset from the LeadingEdge margin. Positive values will shift the image away from the leading edge of the plate while negative values will shift the image toward the leading edge of the plate.

- HomeEdge
  The offset from the HomeEdge margin. Positive values will shift the image away from the home edge of the plate while negative values will shift the image toward the home edge of the plate.

- UnitOfMeasure
  The unit of measure for the LeadingEdge and HomeEdge margins.

3.5.5 Image Clipping

Generally image clipping should be performed by the raster source **because clipping is an expensive operation** – in terms of requiring a significant amount of time and computer resources.

This is an optional field; if it is **not** included when a print session is queued, it is DSI and device specific what is done with image data which is larger than the image area; the print session may be rejected.

If a DSI does not support this capability and it is specified when calling GDAPI_QueuePrintSession or GDAPI_UpdatePrintSessionData, these functions will return the error: GDAPI_NOT_SUPPORTED_IMAGE_CLIPPING.

Data:

```
typedef struct
{
    unsigned long   BytesToClip_FromLeadingEdge;
    unsigned long   BytesToClip_FromTrailingEdge;
    unsigned long   BytesToClip_FromHomeEdge;
    unsigned long   BytesToClip_FromAwayEdge;

} GDAPI_ImageClipping;
```

This allows the raster source to specify the number of bytes to clip from each edge of the image.

Note that the number of bytes to clip may also be dictated by the bit depth of the image data and the type of interleaving used.

3.5.6 Image Data

Image data is specified as part of the print session data. Each image in the print session specifies the following information:

```
typedef struct
{
    GDAPI_MediaSide Side;

    int             numColors;
    GDAPI_Color*    Color;

} GDAPI_Image
```

Related data structures:

```
typedef struct
{
    wchar_t RasterColor[GDAPI_SHORT_STRING];
    wchar_t DeviceColor[GDAPI_SHORT_STRING];

} GDAPI_Color;

typedef enum
{
    MediaSide_Unspecified = IDS_MEDIA_SIDE_UNSPECIFIED,
    MediaSide_Front       = IDS_MEDIA_SIDE_FRONT,
    MediaSide_Back        = IDS_MEDIA_SIDE_BACK

} GDAPI_MediaSide;
```

GDAPI_Image

Field	Description
Side	Perfecting information for each separation; the choices are Front, Back, and Unspecified. This is most critical for proofs which are backed up – in this case the separations in the job which end up on the same piece of media should follow each other in the print session data and the Side information should be set to Front for the first separation and Back for the second.
NumColors	A single image will usually specify one color and will specify two or more colors when the bit depth is greater than 1 and interleaving is not by separation. Multiple colors can be specified for one image if the image data contains information for more than one color – in this case the list of colors specifies the device color names of each color (see the description of GDAPI_Color for a description of device colors).
Color	An array of GDAPI_Color structures; there must be numColors elements in this array.

GDAPI_Color

RasterColor	This required field is the name of the color used by the raster source. There are constants defined for the standard process color names: COLOR_C, COLOR_M, COLOR_Y, and COLOR_K. Other color names, like spot colors, are free form text.
DeviceColor	This optional field, if used, must be one of the device color names returned by the COLOR_NAME attribute. If this is not specified, the DSI will choose a device color based on the RasterColor specified. Proofing media: • Use device color names – to control device settings for the color. • Can require that colors are applied in a specific order (order is determined by COLOR_NAME). In this case the image colors must be sent in the order specified.

Both colors can be specified – to display the mapping between the “raster source color name” and the “device color name”.

Both colors can be specified *after* the call to GDAPI_QueuePrintSession, by calling GDAPI_UpdatePrintSessionData.

3.6 Device State & Sub-state

This section provides an overview of the device state and sub-state reported by the DeviceStateCallback or returned by calling GDAPI_GetDeviceState.

Examples:

Device State	Device Sub-State
Calibrating	GDAPI_SS_CALIBRATING_LASER
OperatorRequired	GDAPI_SS_AIR_PRESSURE_LOW
OperatorRequired	GDAPI_SS_CASSETTE_EMPTY

3.6.1 Device State Overview

The device state is used to report the basic state of the device.

The set of possible device states is fixed.

Device states are not used for print session control, this is provided by PrintSessionCallback.

Some GDAPI functions are not available in some device states. If the device is not in the correct state to handle a given GDAPI function, the function will return a status of GDAPI_INVALID_FOR_DEVICE_STATE; this means that the function is not supported in the current device state.

The GDAPI specification does not specify which GDAPI functions are available during a given device state and which GDAPI functions are not. The only requirement is that a DSI return GDAPI_INVALID_FOR_DEVICE_STATE when a function is not available during a given state.

Note:
If the device state changes during a print session and a GDAPI function returns an error or GDAPI_INVALID_FOR_DEVICE_STATE, then the raster source will abort the current print session. The next synchronization point between the raster source and the DSI will be:

- the queuing of a print session, or
- the release of a print session.

3.6.2 Device Sub-State

The sub-state provides detail for a given device state.

A sub-state code is defined as a GDAPI_Status code with a Facility code of GDAPI_SUB_STATE_FACILITY or any other Facility which is different from GDAPI_FACILITY.

Sub-state codes and function return status codes are both based on a GDAPI_Status code but:

- Device Sub-State codes are to be used only to indicate a Device Sub-State, and
- Return Status codes are to be used only as return status codes.

Code	**Description**
Function Return Status GDAPI_FACILITY	Used only as codes returned as a function status return code.
Device Sub-State GDAPI_SUB_STATE_FACILITY	Used only as a Device Sub-State code.

Sub-state codes are defined by a DSI. Even if two DSIs have the same sub-state code, the codes are interpreted on a per DSI basis.

Sub-state codes can be passed to GDAPI_GetLocalizedString to retrieve a localized string for the sub-state.

3.6.3 Device States

The following are the possible device states:

Device State	Details
Unknown	This state is used by the raster source as the initial state for a device – before the raster source has connected to the device. This state is used by the DSI before a connection has been established to a device or when communications to a device has been lost.
Initializing	The device is powering up.
Calibrating	The device is configuring itself. Examples: re-calibration after many hours of operation.
Offline	An operator is configuring the device. Examples: replenishing media, clearing a jam, servicing.
Idle	The device is configured and capable of imaging.
Imaging	The device is imaging.
LoadingMedia	The device is loading media.
UnloadingMedia	The device is unloading media.
Error	An error occurred; the sub-state will indicate the type of error.
OperatorRequired	Operator interaction is required; the sub-state will indicate why the operator is required.
Shutdown	Indicates that the device is in the process of shutting down.

3.7 Device Attributes Overview

Attributes are used to:

- Determine the capabilities of a device, by asking the DSI which attributes are supported by a device and what the values are for each attribute.
- Select one or more options for an attribute – from a list of options provided for that attribute.
- Track changes in attribute settings initiated by the device.

The basic characteristics of an attribute are:

- An attribute name.
- Data types for reading attribute values and writing attribute values.

Some device attributes are considered "core" and must be supported by a DSI. These are called "Core Attributes" and are listed in the sections which describe core attributes.

Device attributes which are "Non-Core Attributes":

- Have the same basic characteristics as core attributes
- Are supported or not based on device capabilities.

There are some non-core attributes documented in this specification, but non-core attributes should also be documented by a DSI when those attributes differ.

This allows attributes to be handled consistently whether they are "core" or not and allows for the addition, removal, and change of attributes without effecting GDAPI functions which deal with device attributes.

Specific examples of the use of attributes are given in appropriate sections.

3.7.1 Attribute Function Overview

In addition to "core" attribute such as queue length and media, a device may have "non-core" attributes which are specific to the device. The GDAPI provides functions to handle these attributes in a way which allows for change without effecting the GDAPI. Here is an overview of how the attribute functions work together:

Note: if a non-core attribute is changed by the imaging device, the change can be reported via the AttributeChangedCallback mechanism in exactly the same way that changes to core attributes are reported.

Each attribute name is passed as an argument to one of the attribute functions:

- GDAPI_GetNumAvailableAttributeValues
  This call returns a count of the number of values which will be returned by GDAPI_GetAvailableAttributeValue for a given attribute.
- GDAPI_GetAvailableAttributeValue
  This call is used to enumerate the **possible values** of a given attribute.
- GDAPI_GetNumCurrentAttributeValues
  This call returns a count of the number of values which will be returned by GDAPI_GetCurrentAttributeValue for a given attribute.
- GDAPI_GetCurrentAttributeValue
  This function gets the **current value** of an attribute; this may be a sub-set of the possible values returned by GDAPI_GetAvailableAttributeValue.
- GDAPI_SetCurrentAttributeValue
  This function sets the current value of an attribute; this may be a sub-set of the possible values returned by GDAPI_GetAvailableAttributeValue.

The scope of attribute values can be global to a raster source or specific to a print session.

4. SCENARIOS

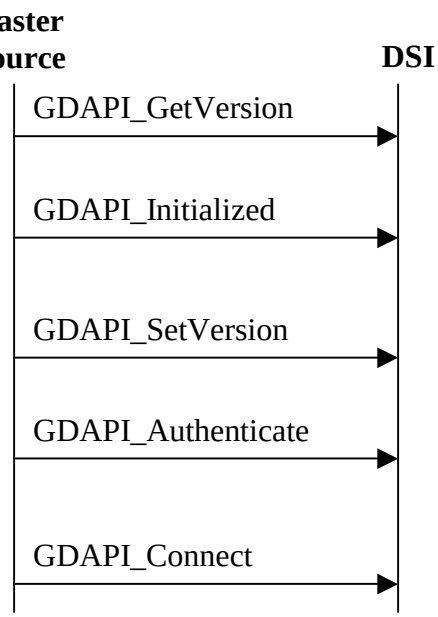
This section presents various scenarios to demonstrate the interaction required between the raster source and a DSI for each of the scenarios presented.

4.1 Startup

4.1.1 Raster Source Initialization

Raster source (RS) initialization involves the raster source connecting to the DSI.

Determine version and connect.



The RS Calls “get version” to determine the range of versions supported by a DSI.

The RS Calls “Initialized” to determine if the GDAPI is initialized; loop on this function until it succeeds.

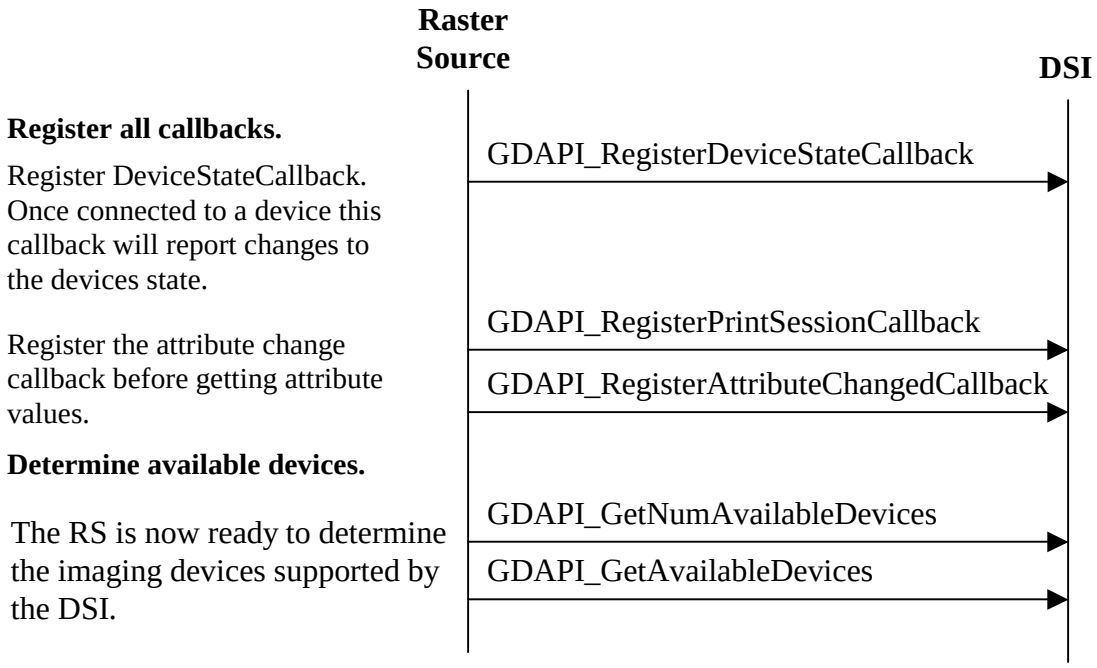
The RS Calls “set version” to establish the version “in common”.

Some DSI require a password to authenticate that the raster source can connect to the GDAPI.

The RS Calls “calls” connect to provide some raster source information to the DSI.

Note that the raster source may want to loop on the call to GDAPI_Initialized to allow the DSI to start. Note also that if a given DSI requires a password before allowing a raster source to connect to the GDAPI that the raster source will not be able to use any of the GDAPI functions (other than GDAPI_GetVersion, GDAPI_Initialized, and GDAPI_SetVersion) until a valid password has been provided.

The next sequence diagram shows the rest of the raster source initialization.



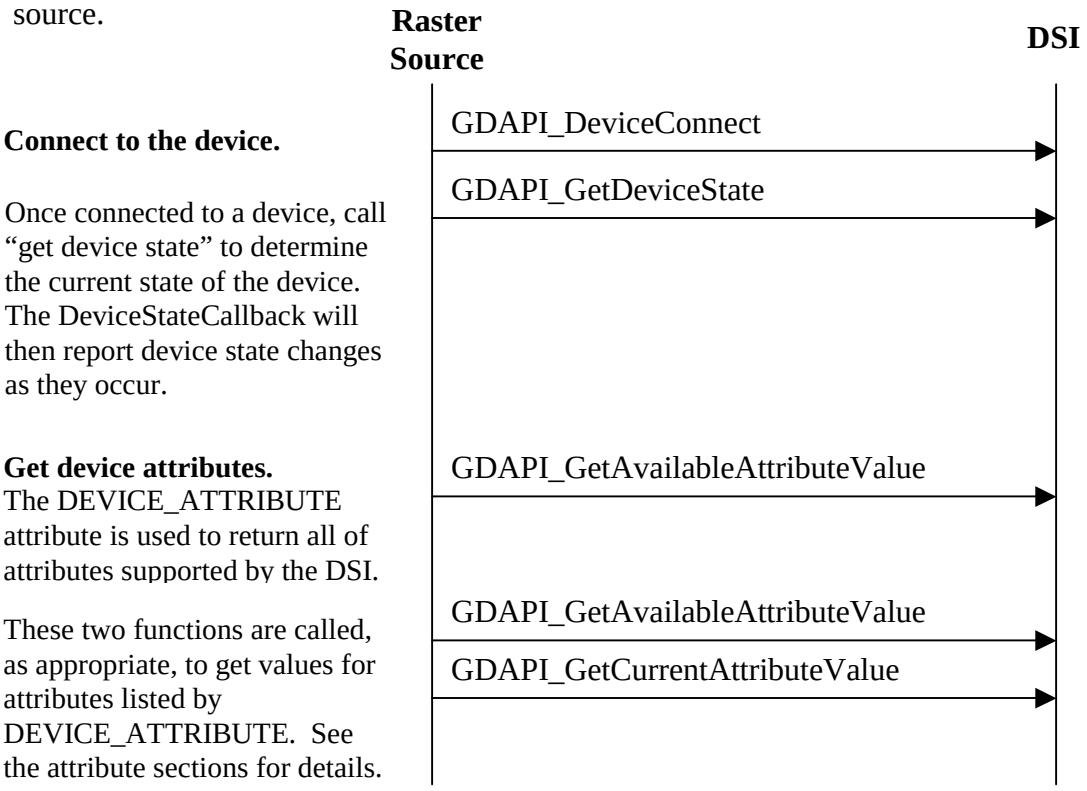
Note that if the raster source cannot find a version which both the raster source and the DSI can support, then the raster source cannot use the DSI.

GDAPI_GetAvailableDevices will report all available output devices, whether or not they are powered on. This call provides basic device information which allows a raster source to select an output device.

Once the raster source has the information about the available output devices, it is ready to connect to one or more devices.

4.1.2 Device Connection

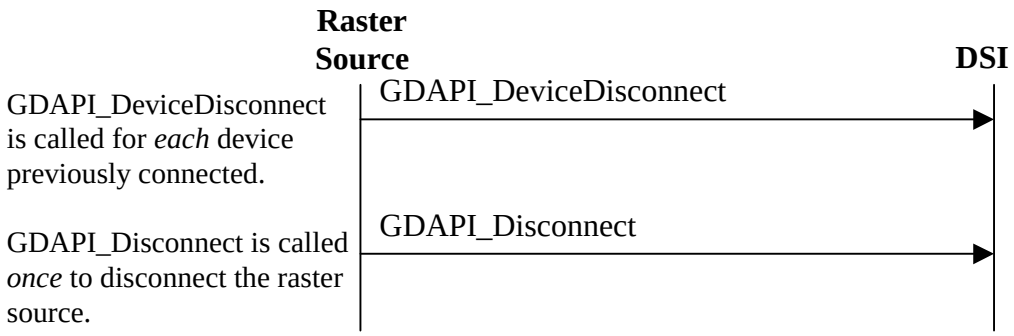
The following calls are made to connect to *each* imaging device used by a raster source.



The device is now ready to queue print sessions.

4.2 Device Disconnect & Raster Source Disconnect

The following calls are made to disconnect a device and the raster source:

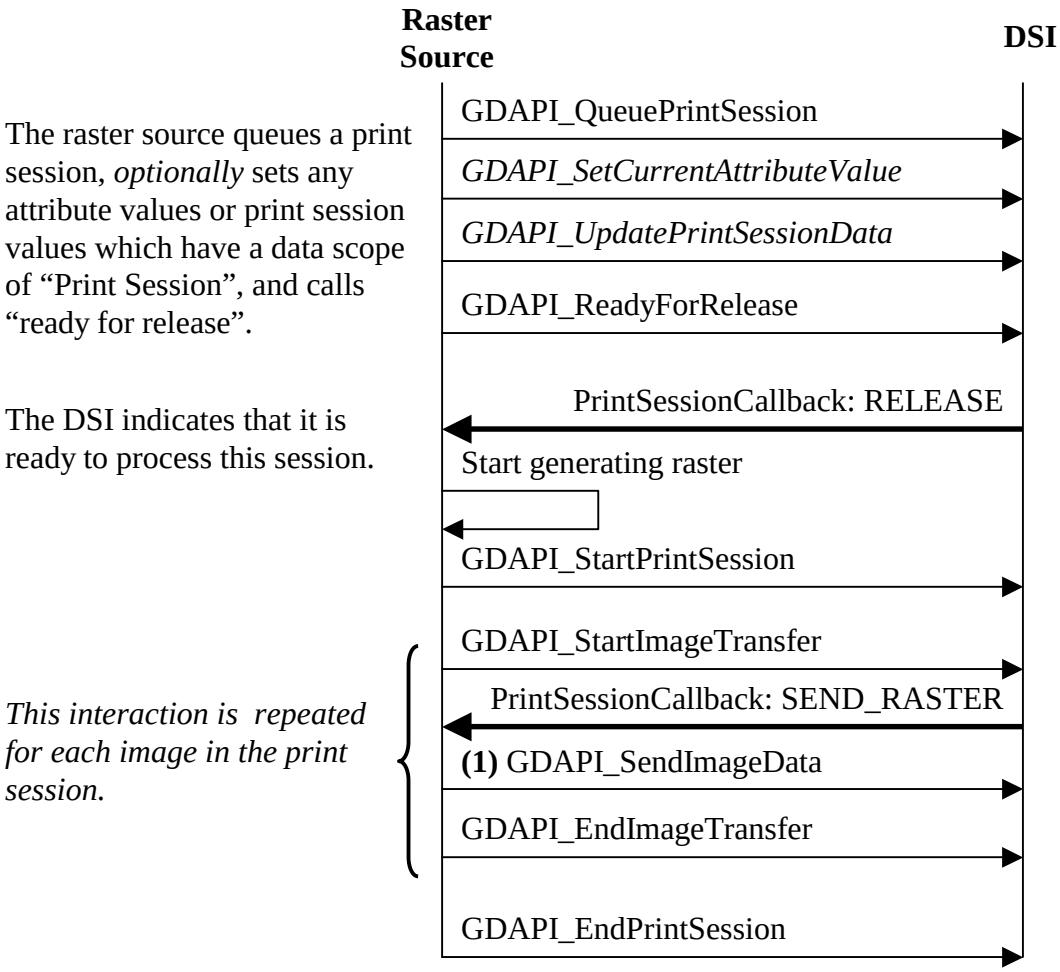


If GDAPI_Disconnect is called and GDAPI_DeviceDisconnect has not been called, then the DSI will disconnect the raster source from each device the raster source was connected to.

4.3 Print Session

Note that print sessions can be released by the DSI in an order which is different from the order in which the print session was queued. If the raster source can not handle a change in the order, it can avoid any re-ordering by queuing print sessions one at a time.

The following interaction occurs to queue and process a print session.

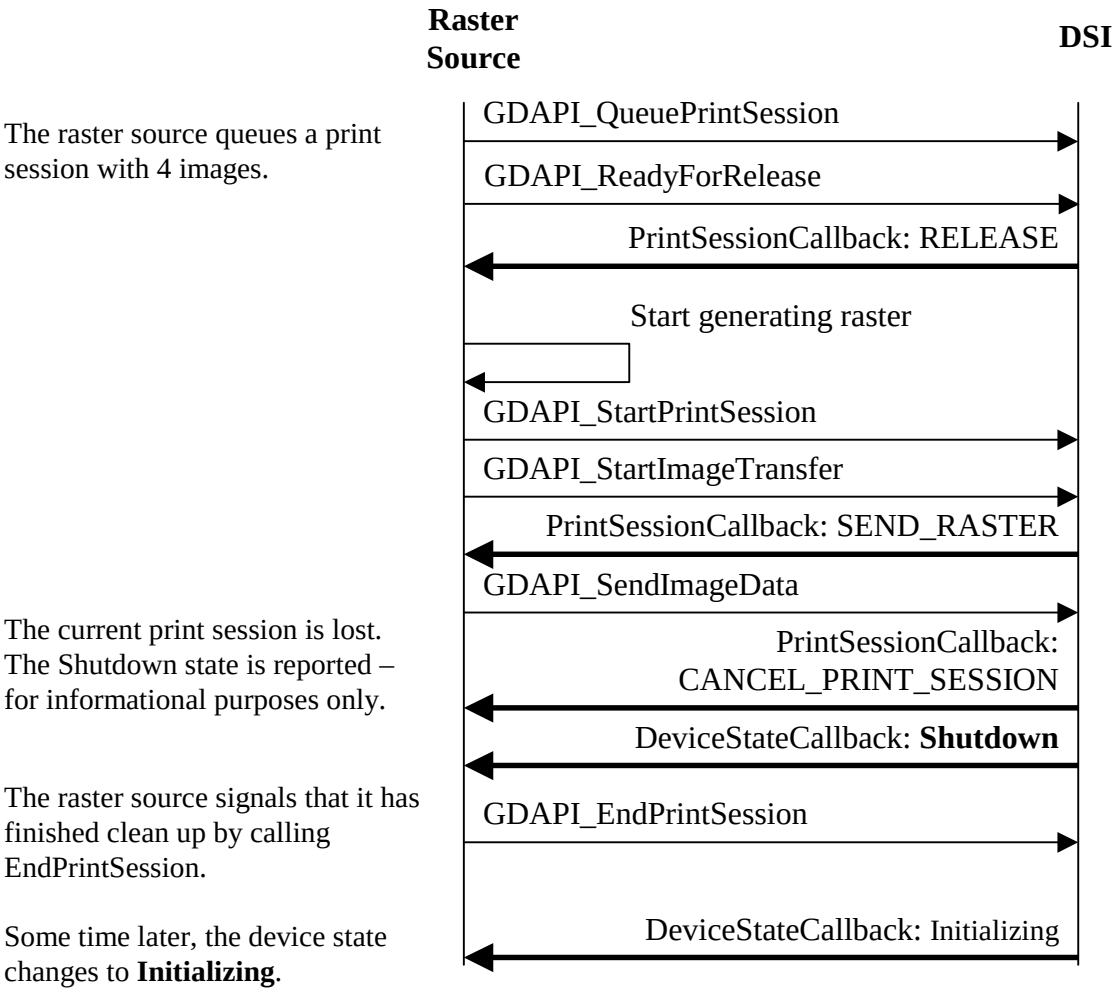


Important points to note are that GDAPI_SendImageData is not called until the DSI sends the SEND_RASTER command via the PrintSessionCallback.

(1) Image data is usually delivered using GDAPI_SendImageData – the documentation of this function describes alternate paths for delivering data.

4.4 Device Shutdown

This scenario shows how a device shutdown can effect the current print session. A DSI may also drop all queued print sessions when the imaging device changes to a Shutdown state, this depends upon the DSI implementation.

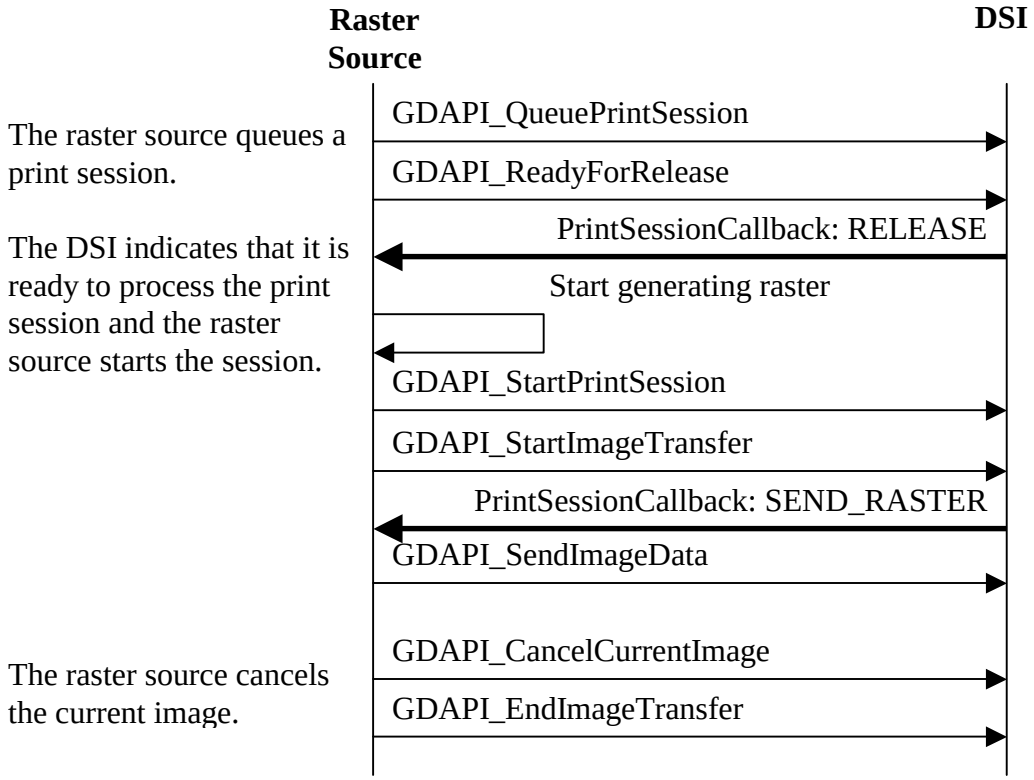


4.5 Cancel Image

An image can be canceled by either the raster source or the imaging device.

4.5.1 Cancel Image: Initiated by the Raster Source

Sequence Diagram:



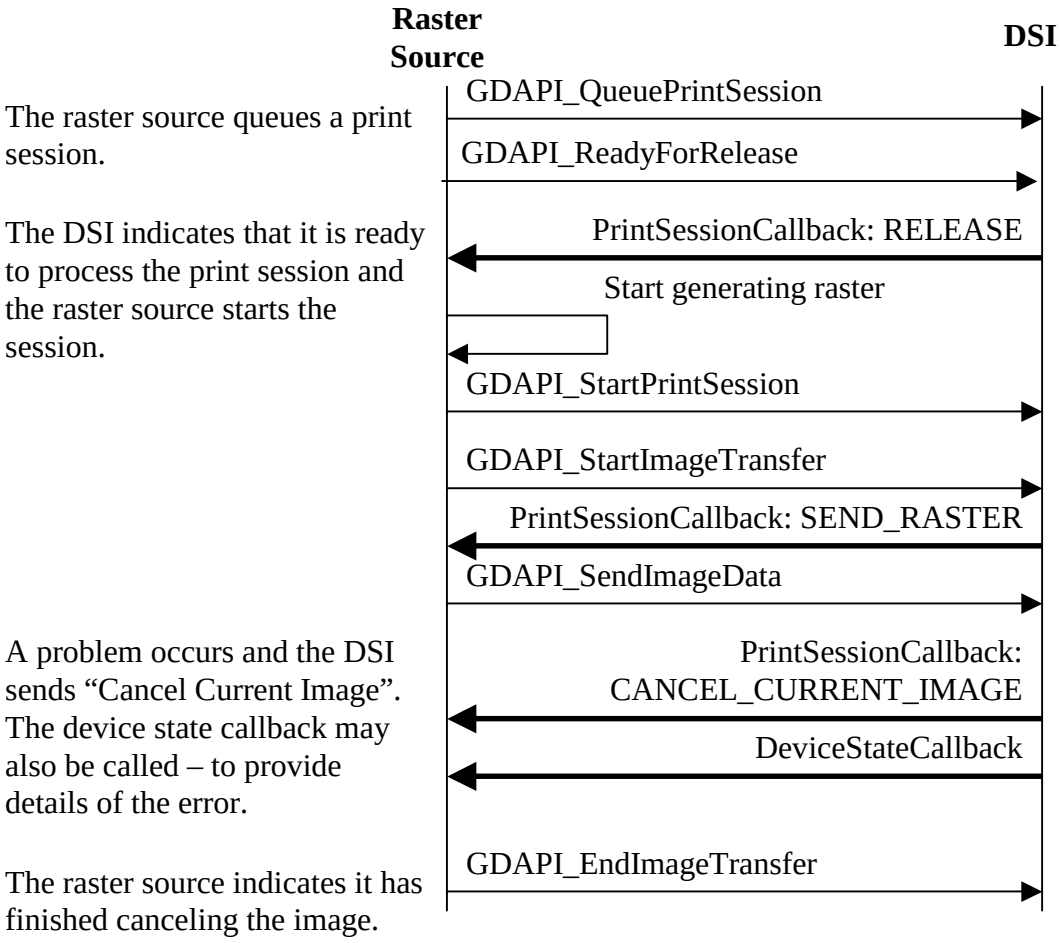
After canceling an image the raster source has the following choices:

- Continue with the next image in the print session by calling GDAPI_StartImageTransfer.
- Cancel the rest of the print session by calling GDAPI_EndPrintSession.

4.5.2 Cancel Image: Initiated by the Imaging Device

An imaging device may need to cancel an image.

Sequence Diagram:



After processing the CANCEL_CURRENT_IMAGE command and calling GDAPI_EndImageTransfer the raster source has the following choices:

- Continue with the next image in the print session by calling GDAPI_StartImageTransfer.
- Cancel the rest of the print session by calling GDAPI_EndPrintSession.

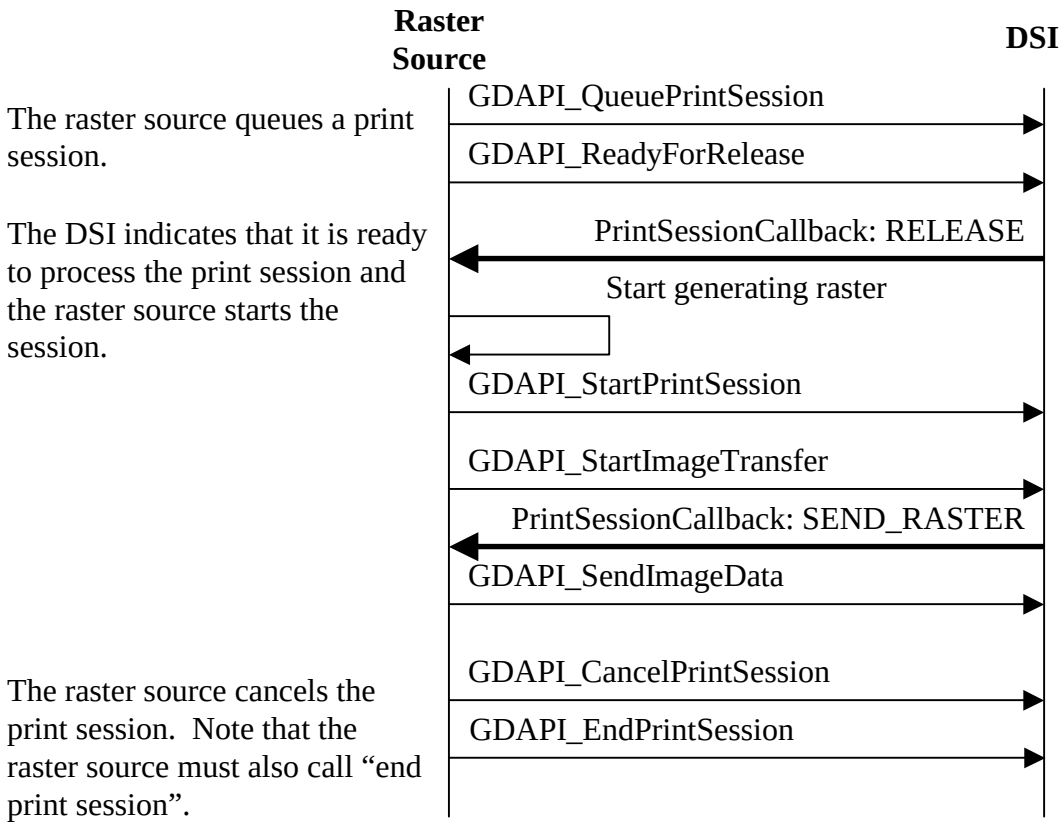
4.6 Cancel Print Session

A print session can be canceled by either the raster source or the imaging device.

4.6.1 Cancel Print Session: Initiated by the raster source

The raster source can choose to abort a print session at any time.

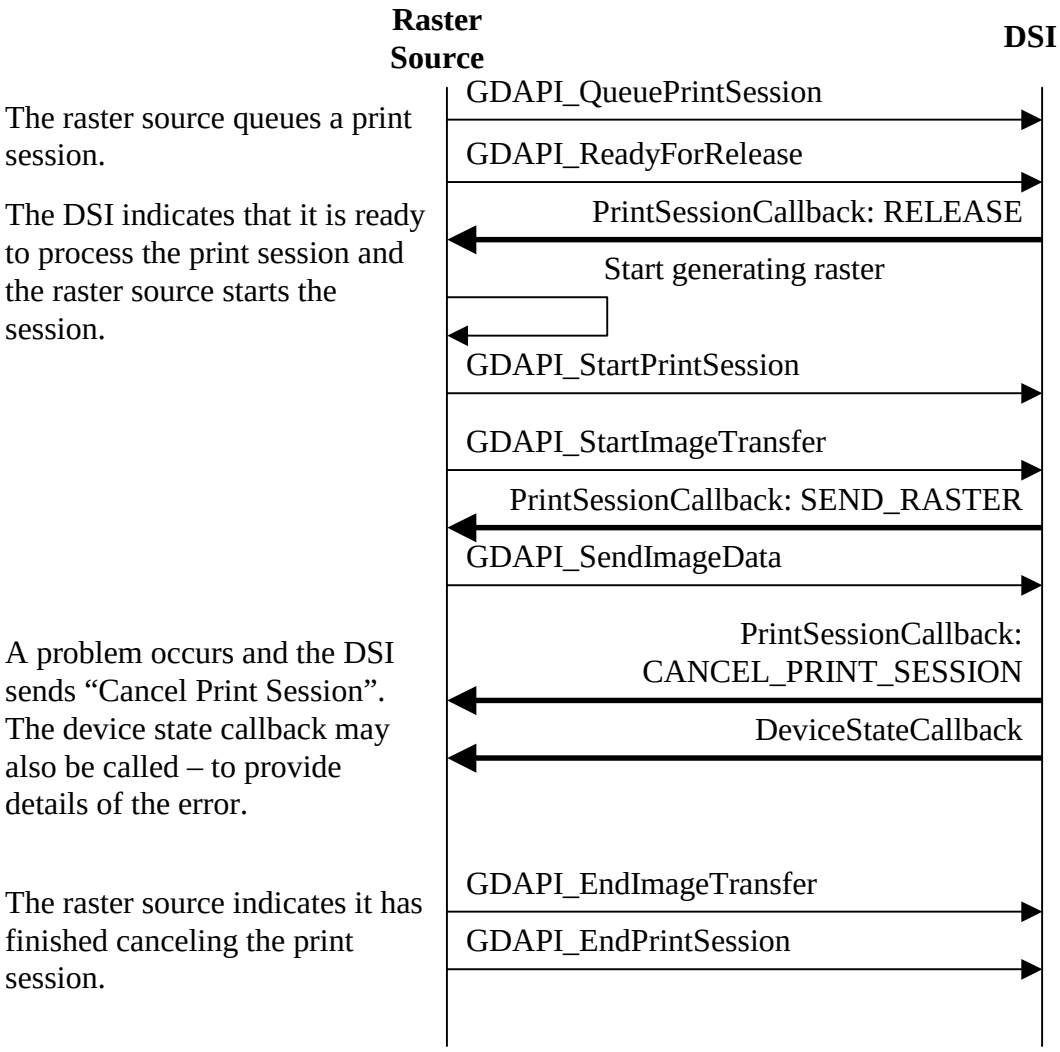
Sequence Diagram:



4.6.2 Cancel Print Session: Initiated by the Imaging device

This can occur when an imaging device is producing a proof from two or more separations and one of the separations is canceled. The proof can only be successfully produced when all separations are present; in this case the imaging device aborts the entire print session.

Sequence Diagram:



5. GDAPI FUNCTIONS

5.1 GDAPI_GetVersion

Description: GDAPI_GetVersion is the first GDAPI call executed by a raster source. The purpose of the call is to determine the range of GDAPI versions supported by the DSI. If the raster source is compatible with the DSI, it calls GDAPI_Initialized to make sure that the GDAPI is initialized and then calls GDAPI_SetVersion to identify the version which the raster source and the DSI have in common.

This approach allows for a new version of a DSI to support older GDAPI versions, and it allows for different logic or different attribute data structures based on version.

Syntax:

GDAPI_Status **GDAPI_GetVersion**(
GDAPI_Version* out_MinVersion,
GDAPI_Version* out_MaxVersion);

Parameters:

out_MinVersion
The lowest GDAPI version that the DSI supports.

out_MaxVersion
The highest GDAPI version that the DSI supports.

Return Code:

- GDAPI_SUCCESS
- GDAPI_FAILURE.

If the GDAPI version used by the raster source falls between the minimum and maximum version supported by the DSI, then the raster source can use the DSI. If GDAPI_GetVersion fails the raster source will not be able to use the DSI.

Related Data:

```
typedef struct
{
    int MajorVersion;
    int MinorVersion;
    int SubVersion;

} GDAPI_Version;
```

5.2 GDAPI_Initialized

Introduced in GDAPI version: 2.0.1

Description: GDAPI_Initialized is called after GDAPI_GetVersion. This function provides an indication of whether or not the DSI implementation of the GDAPI is initialized. Note that if this function returns a failure status that the raster source can use a polling scheme to call this function periodically until the DSI is ready for other GDAPI function calls.

Syntax:

GDAPI_Status **GDAPI_ Initialized** (
GDAPI_Date* out_ServerDate);

Parameters:

out_ServerDate
The date on the server.

Return Code:

- GDAPI_SUCCESS
- A failure status. The DSI is not yet initialized.

Related Data:

```
typedef struct
{
    char Day;
    char Month;
    short Year;

} GDAPI_Date;
```

5.3 GDAPI_SetVersion

Description: GDAPI_SetVersion is called to set the common GDAPI version which the raster source and the DSI will use.

Call GDAPI_GetVersion first to determine the range of GDAPI versions supported. The raster source then sets the “version in common” by calling GDAPI_SetVersion; the "version in common" must be in the range of the versions returned by GDAPI_GetVersion.

Syntax:

GDAPI_Status **GDAPI_SetVersion**(
GDAPI_Version in_GDAPIVersion);

Parameters:

in_GDAPIVersion

The GDAPI_Version structure contains the GDAPI version that is common to the raster source and the DSI.

Return Code:

- GDAPI_SUCCESS
- GDAPI_UNSUPPORTED_VERSION
- GDAPI_FAILURE

If The call to GDAPI_SetVersion fails, the raster source will not be able to use the DSI.

5.4 GDAPI_Authenticate

Introduced in GDAPI version: 2.0.2

Description: The purpose of GDAPI_Authenticate is to authenticate that a given raster source is authorized to connect to the GDAPI.

A DSI may choose not to require authentication, but this should be done by supporting this function and simply returning success no matter what password has been provided.

If a password is required by the DSI, then a valid password must be supplied by a call to GDAPI_Authenticate before the raster source can use any GDAPI functions; *other* than the following functions which can be called before authentication:

- GDAPI_GetVersion
- GDAPI_Initialized
- GDAPI_SetVersion

Syntax:

GDAPI_Status **GDAPI_Authenticate**(
GDAPI_Password in_password);

Parameters:

in_password

If the Device Specific Implementation (DSI) of the GDAPI requires a password, then the raster source must supply a valid password here. If no password is requred, in_password can contain any value of type GDAPI_Password.

Return Code:

- GDAPI_SUCCESS
- GDAPI_INVALID_PASSWORD
  The DSI requires a password and the one supplied was not valid.

Related Data:

```
typedef unsigned long GDAPI_Password;
```

5.5 GDAPI_Connect

Description: The purpose of GDAPI_Connect is to give the DSI some information about the raster source; this is useful to the DSI when more than one raster source uses a given DSI.

Calling GDAPI_Connect also acts as a signal to the DSI to initialize anything required to handle the raster source, which may include initializing connections to available imaging devices.

The DSI must document the raster formats which it supports.

Syntax:

GDAPI_Status **GDAPI_Connect**(
const wchar_t* in_rasterSourceName,
GDAPI_RasterFormat in_rasterFormat);

Parameters:

in_ rasterSourceName
A NULL terminated string which is used by the DSI for information purposes only. This name must be unique for each raster source connected to a given DSI; it is the responsibility of the raster source to make names unique. If the DSI has a GUI, it can display the names of the raster sources currently connected.

in_rasterFormat
The GDAPI_RasterFormat structure describes the format of the raster data that will be supplied by this raster source; see below for details.

Return Code:

If the raster source cannot connect, then it will not be able to use other GDAPI functions.

- GDAPI_SUCCESS
- GDAPI_ALREADY_CONNECTED_TO_GDAPI
  If this function was already called, or a raster source of the same name is already connected.

- GDAPI_DUPLICATE_RASTER_SOURCE_NAME
  Another raster source has already connected using the raster source name specified in in_rasterSourceName.

- GDAPI_UNSUPPORTED_RASTER_FORMAT
  This is returned if in_rasterFormat is not supported by the DSI.

Related Data:

```
typedef struct
{
    short  WordSize;  // the number of bytes per word.
    short  WordOrder; // 0 = big endian, 1 = little endian.
    short  ByteOrder; // 0 = MSBit is first; 1 = MSBit is last.

} GDAPI_RasterFormat;
```

5.6 GDAPI_Disconnect

Description: The purpose of calling GDAPI_Disconnect is to signal the DSI that the raster source is disconnecting and that the DSI can release any resources, such as registered callbacks, or queued print sessions associated with the raster source.

If the raster source is connected to any devices, it should call GDAPI_DeviceDisconnect for each device connection before calling GDAPI_Disconnect. If GDAPI_DeviceDisconnect is not called, then the DSI will disconnect the raster source from each device to which it was connected.

Syntax:
GDAPI_Status GDAPI_Disconnect();

Parameters:
None.

Return Code:

- GDAPI_SUCCESS

- GDAPI_RASTER_SOURCE_NOT_CONNECTED
  GDAPI_Connect has not been called; I.E. the raster source was not connected to the DSI.

5.7 GDAPI_GetNumAvailableDevices

Description: GDAPI_GetNumAvailableDevices is called to determine the number of output devices that are available via this DSI.

Use this function in conjunction with GDAPI_GetAvailableDevices which is used to determine the details of each device.

Syntax:

GDAPI_Status **GDAPI_GetNumAvailableDevices**(
int* out_numDevices);

Parameters:

out_numDevices
The number of output devices available.

Return Code:

- GDAPI_SUCCESS
- GDAPI_FAILURE
- GDAPI_RASTER_SOURCE_NOT_CONNECTED
  GDAPI_Connect has not been called; I.E. the raster source was not connected to the DSI.

5.8 GDAPI_GetAvailableDevices

Description: GDAPI_GetAvailableDevices is called to determine information regarding the output devices that are available via this DSI.

The DSI will report a device even when the device is shut down. Any change in state for such a device will be reported by DeviceStateCallback.

Syntax:

GDAPI_Status **GDAPI_GetAvailableDevices**(
int in_index,
GDAPI_DeviceInfo* out_deviceInfo);

Parameters:

in_index
A 0-based index. This is used to iterate over the available devices. The maximum index is determined by GDAPI_GetNumAvailableDevices.

out_deviceInfo
A pointer to a GDAPI_DeviceInfo structure allocated by the raster source; see below for details.

Return Code:

- GDAPI_SUCCESS
- GDAPI_END_OF_LIST
  in_attributeIndex is one beyond the number of values available for in_attributeName.

- GDAPI_FAILURE
- GDAPI_RASTER_SOURCE_NOT_CONNECTED
  GDAPI_Connect has not been called; I.E. the raster source was not connected to the DSI.

Related Data:

```
typedef struct
{
    GDAPI_DeviceId DeviceId;

    wchar_t  Name[GDAPI_SHORT_STRING];
    wchar_t  Manufacturer[GDAPI_SHORT_STRING];
    wchar_t  Model[GDAPI_SHORT_STRING];
    wchar_t  SerialNumber[GDAPI_SHORT_STRING];
    wchar_t  ServerName[GDAPI_SHORT_STRING];

} GDAPI_DeviceInfo;
```

Notes:

- The GDAPI_DeviceInfo structure contains information that the raster source will use to identify an output device:
  - **DeviceId is used in other GDAPI function calls to refer to the device.** Note that the DSI must ensure that a given raster source gets a unique device ID for each device.
  - The name field provides a name for the device. This may be configurable in the DSI to distinguish between machines which have the same Manufacturer and Model names.
  - ServerName
    A NULL terminated string which indicates the name of a Device Server. A Device Server may serve one or more imaging devices.

5.9 GDAPI_RegisterDeviceStateCallback

Description: GDAPI_RegisterDeviceStateCallback is called by the raster source to register a state callback function with the DSI.

The callback will be called whenever the device changes state or sub-state. Call GDAPI_GetDeviceState to get the current state of the device.
GDAPI_GetDeviceState can also be used as an alternative to the DeviceStateCallback function if a raster source would rather poll for the device state.

Syntax:

GDAPI_Status **GDAPI_RegisterDeviceStateCallback**(
    GDAPI_DeviceStateCallback in_callbackFunction);

Parameters:

in_callbackFunction

Pointer to a 'device status callback' function, which the output device uses to inform the raster source of significant status events. See the "DeviceStateCallback" section for callback details.

Return Code:

- GDAPI_SUCCESS
- GDAPI_FAILURE

Notes:

- The format and parameters of the *callback* function are described in the DeviceStateCallback section.

5.10 DeviceStateCallback

Description: DeviceStateCallback is a callback that is registered via the GDAPI_RegisterDeviceStateCallback function. It is invoked by the output device to report a change in the device state or sub-state.

Another method of getting the device state and sub-state is to use the GDAPI_GetDeviceState function.

The raster source should not perform any significant operations in the callback; it should note the information and consume it in its own operational loop. This implies that none of the GDAPI calls should be executed in the callback.

This reporting mechanism is:

- Not for print session control, which is provided by PrintSessionCallback.
- For informational purposes only.
- Allows for DSI specific sub-states.

DeviceStateCallback will be called when either one or both of the following occur:

- The device changes state.
- The DSI wants to report a sub-state.

Syntax:

GDAPI_Status **DeviceStateCallback**(
GDAPI_DeviceId in_deviceId,
GDAPI_DeviceState in_deviceState,
GDAPI_Status in_deviceSubState);

Parameters:

in_deviceId
: The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_deviceState
: The current state of the output device; see the “Device States” section for details. This indicates the basic state of the device.

in_deviceSubState
: A GDAPI_Status code describing the device sub-state. See the “Device Sub-State” section for details. The sub-state is used to report detail regarding the current device state; for example, State: Calibrating, Sub-State: GDAPI_CALIBRATING_LASER.

Return Code:

- GDAPI_SUCCESS.
- GDAPI_UNKNOWN_DEVICE_ID if the Raster Source has not connected to the device ID specified.

In the event of a failure, the output device should report an error to the operator (using its UI if available).

Notes:

Use GDAPI_GetMessage to retrieve a textual representation of in_deviceSubState.

5.11 GDAPI_GetDeviceState

Description: The GDAPI_GetDeviceState function is used to get the current device state and sub-state.

Another method of getting the device state and sub-state is to use the GDAPI_RegisterDeviceStateCallback function.

Syntax:

GDAPI_Status **GDAPI_GetDeviceState**(
GDAPI_DeviceId in_deviceId,
GDAPI_DeviceState* out_deviceState,
GDAPI_Status* out_deviceSubState);

Parameters:

in_deviceId
: The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

out_deviceState
: The current state of the output device; see the “Device States” section for details. This indicates the basic state of the device.

out_deviceSubState

A GDAPI_Status code describing the device sub-state. See the “Device Sub-State” section for details. The sub-state is used to report detail regarding the current device state; for example, State: Calibrating, Sub-State: GDAPI_CALIBRATING_LASER.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_FAILURE

Notes:

Use GDAPI_GetMessage to retrieve a textual representation of in_deviceSubState.

5.12 GDAPI_RegisterPrintSessionCallback

Description: This function is called by the raster source to register a “print session control” callback function with the DSI. The callback function is then used by the DSI to send “print session control” commands; See PrintSessionCallback for details.

Syntax:

GDAPI_Status **GDAPI_RegisterPrintSessionCallback**(
GDAPI_PrintSessionCallback in_callbackFunction);

Parameters:

in_callbackFunction

A pointer to the PrintSessionCallback function. The format and parameters of the callback function are described in the section below entitled ‘PrintSessionCallback’.

Return Code:

- GDAPI_SUCCESS
- GDAPI_FAILURE

5.13 PrintSessionCallback

Description: The PrintSessionCallback provides print session control for a print session which has been queued by GDAPI_QueuePrintSession. When

PrintSessionCallback is called, the raster source uses the device and session IDs to look up a queued session and then process the session according to the value of the session command.

The raster source should not perform any significant operations in the callback; it should note the information and consume it in its own operational loop. This implies that none of the GDAPI calls should be executed in the callback.

The DSI is required to implement this as a broadcast mechanism, I.E. the DSI will call all registered PrintSessionCallback functions whenever the DSI reports a change to the queued print sessions. This will act as a crude progress indicator to any raster source which is not handling the current print session.

Syntax:

GDAPI_Status **PrintSessionCallback**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId,
GDAPI_PrintSessionCommand in_printSessionCommand);

Parameters:

in_deviceId
The ID that identifies the output device; this is a device ID previously specified to a GDAPI_QueuePrintSession call.

in_printSessionId
The ID to identify the session; originally supplied by GDAPI_QueuePrintSession.

in_printSessionCommand
This enumerated type specifies a command for in_printSessionId. See below for details of the possible commands.

Return Code:

- GDAPI_SUCCESS
  to indicate the raster source will process the print session according to in_SessionCommand.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  to indicate that the DSI has invoked PrintSessionCallback for a raster source which does not handle this print session.
- GDAPI_REQUEUE_PRINT_SESSION
  to indicate that the raster source is not ready to handle a given print session and that the DSI can process another print session before returning to this one.

Print Session Commands:

Command	Description

RELEASE	The DSI sends this when it is ready to commit to outputting a job. The raster source should start generating raster, but should not send raster until the DSI requests it. The DSI will initiate media loading after a successful return from the callback issuing the RELEASE command.
SEND_RASTER	The device is ready to receive raster for a separation, I.E. media has been loaded and the device is ready. **This command is sent once for each separation in a print session.**
CANCEL_PRINT_SESSION	The raster source should cancel the print session regardless of the print session state. The raster source **must** call GDAPI_EndPrintSession to signal it has finished canceling a print session.
CANCEL_CURRENT_IMAGE	The raster source should cancel the current image. The raster source calls GDAPI_EndImageTransfer to signal it has finished canceling an image.

5.14 GDAPI_RegisterAttributeChangedCallback

Description: The raster source calls this function to register a callback which will notify the raster source of configuration changes on the output device, for example: changes in the media which are currently available.

Syntax:

GDAPI_Status **GDAPI_RegisterAttributeChangedCallback**(
GDAPI_AttributeChangedCallback in_callbackFunction);

Parameters:

in_callbackFunction

A pointer to a “attribute changed callback” function, which the DSI uses to inform the raster source of attribute changes. The format and parameters of the callback function are described in the AttributeChangedCallback section

Return Code:

- GDAPI_SUCCESS
- GDAPI_FAILURE.

5.15 AttributeChangedCallback

Description: AttributeChangedCallback is a callback that is invoked by the DSI when a configuration change occurs on the device.

The callback can signal a change to either “Core” or “Non-Core” Attributes.

The raster source should not perform any significant operations in the callback; it should note the information and consume it in its own operational loop. This implies that none of the GDAPI calls should be executed in the callback.

Syntax:

GDAPI_Status **AttributeChangedCallback**(
GDAPI_DeviceId in_deviceId,
const wchar_t* in_changedAttribute);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_changedAttribute
The NULL terminated name of the attribute that has changed.

Return Code:

- GDAPI_SUCCESS.
- GDAPI_UNKNOWN_DEVICE_ID if the Raster Source has not connected to the device ID specified.
- GDAPI_UNKNOWN_ATTRIBUTE if the attribute specified is not known.

In the event of a failure, the DSI should report an error to the operator (using its UI if available).

Notes:

- The callback says which attribute has changed. The in_changedAttribute argument will tell the raster source the name of the attribute that has changed; this name can then be used in GDAPI_GetAvailableAttributeValue to determine the new value of the attribute. The callback will be called for each attribute that has changed.
- The raster source should propagate the new configuration information as required.

5.16 GDAPI_GetNumAvailableAttributeValues

Description: GDAPI_GetNumAvailableAttributeValues gets a count of the number of values which will be returned by GDAPI_GetAvailableAttributeValue for a given attribute.

This can be called before calling GDAPI_GetAvailableAttributeValue to help the raster source determine how many elements to allocate for a given attribute. It does

not *need* to be called as GDAPI_GetAvailableAttributeValue returns one attribute value per call and returns GDAPI_END_OF_LIST when there are no more values.

It is the responsibility of each DSI to document the attributes it supports.

Syntax:

GDAPI_Status **GDAPI_GetNumAvailableAttributeValues**(
GDAPI_DeviceId in_deviceId,
const wchar_t* in_attributeName,
int* out_attributeCount);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_attributeName
A NULL terminated string which the caller sets to the name of the attribute; GDAPI_GetNumAvailableAttributeValues returns the number of values which will be returned by GDAPI_GetAvailableAttributeValue for this attribute.

out_attributeCount
The count of the number of values which will be returned by GDAPI_GetAvailableAttributeValue for in_attributeName. This may be GDAPI_NO_VALUE for an attribute which does not have a set number of possible values.

Return Value:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_ATTRIBUTE
  in_attributeName specified an unknown attribute.
- GDAPI_INVALID_ATTRIBUTE_FOR_DEVICE
  in_attributeName is not supported by the device.

5.17 GDAPI_GetAvailableAttributeValue

Description: GDAPI_GetAvailableAttributeValue gets the possible values for an attribute; each call returns one of the possible values.

Available values can be changed by a DSI but are always read only for a raster source.

It is the responsibility of each DSI to document the attributes it supports.

Syntax:

GDAPI_Status **GDAPI_GetAvailableAttributeValue** (
GDAPI_DeviceId in_deviceId,
const wchar_t* in_attributeName,
int in_attributeIndex,
int in_numBytes,
void* out_attribute);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_attributeName
A NULL terminated string which the caller sets to the name of the attribute.

in_attributeIndex
The 0-based index to the i^{th} value of in_attributeName. The maximum value for this index can be determined by GDAPI_GetNumAvailableAttributeValues or the caller can increment the index until GDAPI_GetAvailableAttributeValue returns GDAPI_END_OF_LIST.

in_numBytes
The expected size, in bytes, of out_attribute.

out_attribute
This is a pointer to the basic type or structure which is appropriate for in_attributeName; the pointer must be cast to a void pointer. The storage pointed to is allocated by the raster source.

Return Value:

- GDAPI_SUCCESS
  One possible attribute value will be in out_attribute.
- GDAPI_END_OF_LIST
  in_attributeIndex is one beyond the number of values available for in_attributeName.
- GDAPI_UNKNOWN_ATTRIBUTE
  in_attributeName specified an unknown attribute.
- GDAPI_INVALID_ATTRIBUTE_FOR_DEVICE
  in_attributeName is not supported by the device.

- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_INVALID_SIZE
  in_numBytes specified an invalid size for this attribute.

5.18 GDAPI_GetNumDefaultAttributeValues

Introduced in GDAPI version: 2.0.3

Description: GDAPI_GetNumDefaultAttributeValues gets a count of the number of values that will be returned by GDAPI_GetDefaultAttributeValue for a given attribute.

This can be called before calling GDAPI_GetDefaultAttributeValue to help the raster source determine how many elements to allocate for a given attribute. It does not *need* to be called as GDAPI_GetDefaultAttributeValue returns one attribute value per call and returns GDAPI_END_OF_LIST when there are no more values.

It is the responsibility of each DSI to document the attributes it supports.

Syntax:

GDAPI_Status **GDAPI_GetNumDefaultAttributeValues**(
GDAPI_DeviceId in_deviceId,
const wchar_t* in_attributeName,
int* out_attributeCount);

Parameters:

in_deviceId
: The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_attributeName
: A NULL terminated string which the caller sets to the name of the attribute; GDAPI_GetNumCurrentAttributeValues returns the number of values which will be returned by GDAPI_GetCurrentAttributeValue for this attribute.

out_attributeCount
: The count of the number of values which will be returned by GDAPI_GetDefaultAttributeValue for in_attributeName.

Return Value:

- GDAPI_SUCCESS

- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_ATTRIBUTE
  in_attributeName specified an unknown attribute.
- GDAPI_INVALID_ATTRIBUTE_FOR_DEVICE
  in_attributeName is not supported by the device.

5.19 GDAPI_GetDefaultAttributeValue

Introduced in GDAPI version: 2.0.3

Description: GDAPI_GetDefaultAttributeValue gets the default value of an attribute.

An attribute may have no default value, a single default value or multiple default values – if the attribute represents a list.

It is the responsibility of each DSI to document the attributes it supports.

Syntax:

GDAPI_Status **GDAPI_GetDefaultAttributeValue**(
GDAPI_DeviceId in_deviceId,
const wchar_t* in_attributeName,
int in_attributeIndex,
int in_numBytes,
void* out_attribute);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_attributeName
A NULL terminated string which the caller sets to the name of the attribute; GDAPI_GetDefaultAttributeValue will return the value of this attribute.

in_attributeIndex
The 0-based index to the i^{th} default value of in_attributeName. The maximum value for this index can be determined by calling GDAPI_GetNumDefaultAttributeValues or the caller can increment the index until GDAPI_GetDefaultAttributeValue returns GDAPI_END_OF_LIST

in_numBytes
The size, in bytes, of out_attribute.

out_attribute

This is a pointer to the basic type or structure which is appropriate for in_attributeName; the pointer must be cast to a void pointer. The storage pointed to is allocated by the raster source.

Return Value:

- GDAPI_SUCCESS
  The current value of the attribute is in out_attribute.
- GDAPI_END_OF_LIST
  in_attributeIndex is one beyond the number of values available for in_attributeName.
- GDAPI_UNKNOWN_ATTRIBUTE
  in_attributeName specified an unknown attribute.
- GDAPI_INVALID_ATTRIBUTE_FOR_DEVICE
  in_attributeName is not supported by the device.
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_INVALID_SIZE
  in_numBytes specified an invalid size for this attribute.
- GDAPI_NO_DEFAULT_ATTRIBUTE_VALUE
  This attribute does not have a default value.

5.20 GDAPI_GetNumCurrentAttributeValues

Description: GDAPI_GetNumCurrentAttributeValues gets a count of the number of values that will be returned by GDAPI_GetCurrentAttributeValue for a given attribute.

This can be called before calling GDAPI_GetCurrentAttributeValue to help the raster source determine how many elements to allocate for a given attribute. It does not *need* to be called as GDAPI_GetCurrentAttributeValue returns one attribute value per call and returns GDAPI_END_OF_LIST when there are no more values.

It is the responsibility of each DSI to document the attributes it supports.

Syntax:

GDAPI_Status **GDAPI_GetNumCurrentAttributeValues**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId,

GDAPI_SeparationId in_separationId,
const wchar_t* in_attributeName,
int* out_attributeCount);

Parameters:

in_deviceId
: The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
: The ID to identify the session; supplied by GDAPI_QueuePrintSession. This can be GDAPI_NO_VALUE for a global value.

in_separationId
: The 0-based index to identify a specific separation, or GDAPI_NO_VALUE for an attribute value that applies to all separations in a print session. Note that **in_printSessionId** can not be GDAPI_NO_VALUE if in_separationId specifies a specific separation.

in_attributeName
: A NULL terminated string which the caller sets to the name of the attribute; GDAPI_GetNumCurrentAttributeValues returns the number of values which will be returned by GDAPI_GetCurrentAttributeValue for this attribute.

out_attributeCount
: The count of the number of values which will be returned by GDAPI_GetCurrentAttributeValue for in_attributeName.

Return Value:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_UNKNOWN_ATTRIBUTE
  in_attributeName specified an unknown attribute.
- GDAPI_INVALID_ATTRIBUTE_FOR_DEVICE
  in_attributeName is not supported by the device.

5.21 GDAPI_GetCurrentAttributeValue

Description: GDAPI_GetCurrentAttributeValue gets the current value of an attribute.

The current value of an attribute is:

- Initialized by a DSI; the initial value may be the empty set or a default value.
- Changed by calling GDAPI_SetCurrentAttributeValue.

A current attribute can have a single value or multiple values if the attribute represents a list.

It is the responsibility of each DSI to document the attributes it supports.

Syntax:

GDAPI_Status **GDAPI_GetCurrentAttributeValue**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId,
GDAPI_SeparationId in_separationId,
const wchar_t* in_attributeName,
int in_attributeIndex,
int in_numBytes,
void* out_attribute);

Parameters:

in_deviceId
: The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
: The ID to identify the session; supplied by GDAPI_QueuePrintSession. This can be GDAPI_NO_VALUE for a global value.

in_separationId
: The 0-based index to identify a specific separation, or GDAPI_NO_VALUE for an attribute value that applies to all separations in a print session. Note that **in_printSessionId** can not be GDAPI_NO_VALUE if in_separationId specifies a specific separation.

in_attributeName
: A NULL terminated string which the caller sets to the name of the attribute; GDAPI_GetCurrentAttributeValue will return the value of this attribute.

in_attributeIndex
: The 0-based index to the i^{th} value of in_attributeName. The maximum value for this index can be determined by GDAPI_GetNumCurrentAttributeValues or the caller can increment the index until GDAPI_GetCurrentAttributeValue returns GDAPI_END_OF_LIST

in_numBytes
The size, in bytes, of out_attribute.

out_attribute
This is a pointer to the basic type or structure which is appropriate for in_attributeName; the pointer must be cast to a void pointer. The storage pointed to is allocated by the raster source.

Return Value:

- GDAPI_SUCCESS
  The current value of the attribute is in out_attribute.
- GDAPI_UNKNOWN_ATTRIBUTE
  in_attributeName specified an unknown attribute.
- GDAPI_INVALID_ATTRIBUTE_FOR_DEVICE
  in_attributeName is not supported by the device.
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_INVALID_SIZE
  in_numBytes specified an invalid size for this attribute.
- GDAPI_NO_ATTRIBUTE_VALUE
  This can be returned if in_printSessionId was GDAPI_NO_VALUE and there is no global value set for this attribute, or if in_printSessionId was specified and there is no global value or print session value set for this attribute.

5.22 GDAPI_SetCurrentAttributeValue

Description: GDAPI_SetCurrentAttributeValue sets the current value of an attribute.

See the section “Data Scope” for a description of when attribute values can be set.

Note that the scope of a current value is either global to a raster source, specific to a print session, or specific to a separation

A current attribute value comes into scope:

- After the first GDAPI_StartPrintSession call following a call to GDAPI_SetCurrentAttributeValue.

A current attribute value goes out of scope:

- For global values – when any of the following are called: GDAPI_Disconnect, GDAPI_DeviceDisconnect.
- For values specific to a print session – when GDAPI_EndPrintSession is called for the print session associated with the value.
- For values specific to a separation – when GDAPI_EndImageTransfer is called for the separation associated with the value.

Syntax:

GDAPI_Status **GDAPI_SetCurrentAttributeValue**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId,
GDAPI_SeparationId in_separationId,
const wchar_t* in_attributeName,
int in_numBytes,
void* in_attribute);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession. This can be GDAPI_NO_VALUE for global attributes.

in_separationId
The 0-based index to identify a specific separation, or GDAPI_NO_VALUE for an attribute value that applies to all separations in a print session. Note that **in_printSessionId** can not be GDAPI_NO_VALUE if in_separationId specifies a specific separation.

in_attributeName
A NULL terminated string which the caller sets to the name of the attribute; GDAPI_SetCurrentAttributeValue will set the value of this attribute.

in_numBytes
The size, in bytes, of in_attribute.

in_attribute
This is a pointer to the basic type or structure which is appropriate for in_attributeName; the pointer must be cast to a void pointer. The storage pointed to is allocated by the raster source.

Return Value:

- GDAPI_SUCCESS
  The current attribute value was set.
- GDAPI_UNKNOWN_ATTRIBUTE
  in_attributeName specified an unknown attribute.
- GDAPI_INVALID_ATTRIBUTE_FOR_DEVICE
  in_attributeName is not supported by the device.
- GDAPI_READ_ONLY
  This attribute can be read, using GDAPI_GetCurrentAttributeValue, but **cannot** be written.
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_INVALID_SIZE
  in_numBytes specified an invalid size for this attribute.
- GDAPI_OUT_OF_SCOPE
  See the “Data Scope” section for details of how this can happen.

5.23 GDAPI_DeleteCurrentAttributeValue

Introduced in GDAPI version: 2.0.3

Description: GDAPI_DeleteCurrentAttributeValue deletes the current value of an attribute.

Current attribute values are deleted when a print session finishes and when a raster source disconnects – the raster source does not normally need to explicitly delete attribute values.

The raster source can delete a current attribute value from one scope to pick up the value from a larger scope. Note that the scope of a current value is either global to a raster source, specific to a print session, or specific to a separation.

Example:

The following values have been set for the INVERT_RASTER attribute:

- A ‘default print session value’ of ‘true’.
- A value specific to separation 2 of ‘false’.

Deleting the attribute value specific to separation 2 will mean that separation 2 will use the default value set for the print session. This is more appropriate than setting the INVERT_RASTER value for separation 2 to 'true' as it means that separation 2 will take the default value, even if it is later changed.

Syntax:

GDAPI_Status **GDAPI_DeleteCurrentAttributeValue**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId,
GDAPI_SeparationId in_separationId,
const wchar_t* in_attributeName);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession. This can be GDAPI_NO_VALUE for global attributes.

in_separationId
The 0-based index to identify a specific separation, or GDAPI_NO_VALUE for an attribute value that applies to all separations in a print session. Note that **in_printSessionId** can not be GDAPI_NO_VALUE if in_separationId specifies a specific separation.

in_attributeName
A NULL terminated string which the caller sets to the name of the attribute; GDAPI_DeleteCurrentAttributeValue will delete the value of this attribute.

Return Value:

- GDAPI_SUCCESS
  The current attribute value was deleted.
- GDAPI_UNKNOWN_ATTRIBUTE
  in_attributeName specified an unknown attribute.
- GDAPI_INVALID_ATTRIBUTE_FOR_DEVICE
  in_attributeName is not supported by the device.
- GDAPI_READ_ONLY
  This attribute can be read, using GDAPI_GetCurrentAttributeValue, but **cannot** be deleted.
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.

- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_NO_ATTRIBUTE_VALUE
  There was no current attribute value as specified by the input parameters.

5.24 GDAPI_DeviceConnect

Description: GDAPI_DeviceConnect is used by the raster source to request a connection to an output device reported by GDAPI_GetAvailableDevices.

Syntax:

GDAPI_Status GDAPI_DeviceConnect(
GDAPI_DeviceId in_deviceId);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

Return Code:

It is expected that if GDAPI_DeviceConnect fails that the raster source will not attempt to send any further commands to the DSI.

- GDAPI_SUCCESS
- GDAPI_UNKNOWN_DEVICE_ID
- GDAPI_ALREADY_CONNECTED_TO_DEVICE
  If the raster source is already connected to this device.
- GDAPI_RASTER_SOURCE_NOT_CONNECTED
  GDAPI_Connect has not been called; I.E. the raster source was not connected to the DSI.

Notes:

- in_deviceId is required to allow a single raster source to simultaneously connect to multiple output devices. Each GDAPI call contains the device ID of the output device that the raster source wishes to communicate with.
- When the DSI receives this call, it should register the raster source so that callbacks will be made to the appropriate raster source.

5.25 GDAPI_DeviceDisconnect

Description: GDAPI_DeviceDisconnect is called when the raster source disconnects from an output device. Calling this function indicates that the raster source will no longer communicate, or send raster, to the output device

The raster source should cancel any print sessions which are queued for this device before disconnecting from the device.

This is the signal that the DSI can perform any required cleanup for this raster source and device. The cleanup can include:

- De-registering callbacks which were registered for this device.
- Canceling any queued print sessions.

Syntax:

GDAPI_Status GDAPI_DeviceDisconnect(
GDAPI_DeviceId in_deviceId);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

Return Code:

A failure during GDAPI_DeviceDisconnect is relatively meaningless to the raster source except that it should display the error message to alert the operator of the GDAPI_DeviceDisconnect failure.

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_FAILURE

5.26 GDAPI_QueuePrintSession

Description: GDAPI_QueuePrintSession is used to queue a print session. This call is used to place an entry into the output device's queue; if successfully queued, the output device will provide a unique print session identifier which is used in other GDAPI print session functions.

Note: The “Print Session” section provides details on print session *control* and *data* which apply directly to the use of this function.

A queued print session will not become active, receive the RELEASE signal from the PrintSessionCallback, until the raster source has called GDAPI_ReadyForRelease.

The raster source should not produce raster until signaled, via PrintSessionCallback, to RELEASE a print session; this is the signal from the DSI that a given print session will be processed next.

Queuing a print session allows an output device to prepare for the print session.

Syntax:

GDAPI_Status **GDAPI_QueuePrintSession**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionData in_printSessionData,
GDAPI_PrintSessionId* out_printSessionId);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionData
This structure defines a given print session; see the “Print Session Data” section for details.

out_printSessionId
The ID to identify the print session; this ID is used to distinguish this session from others. The ID will be 0 or greater.

Return Code:

- GDAPI_SUCCESS
- GDAPI_CALLBACK_NOT_REGISTERED
  The PrintSessionCallback must be registered before calling GDAPI_QueuePrintSession.
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_INVALID_ARGUMENT
  The data in in_printSessionData was not valid.
- GDAPI_QUEUE_FULL
  The print session could not be queued as the queue is full.

- GDAPI_FAILURE

The DSI will never request raster data for a print session that failed to queue; however, the raster source may attempt to queue the session again in the future.

5.27 GDAPI_UpdatePrintSessionData

Description: GDAPI_UpdatePrintSessionData is used to update the print session data, when some of that data is discovered or changed, after a print session has initially been queued.

Note:
The "Print Session" section provides details on print session *control* and *data* which apply directly to the use of this function.

Some uses of this function are to:

- Set the size of the image; the raster source may not know the size of the image when the print session is queued. The image size must be set before the raster data is sent for the first image.
- Change the number of separations; this can be done any time before a print session has finished. The number of separations may change between the time a print session is queued and the time a print session is processed. Example, when the input is coming from a PostScript file, the file can contain an unknown number of data sections followed by ShowPage commands; the raster source does not know how many ShowPage commands the PostScript file contains and can queue the print session before processing the entire PostScript file.

See the section on Print Session Data for a description of when each type of Print Session data can be changed.

Syntax:

GDAPI_Status **GDAPI_UpdatePrintSessionData**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId,
GDAPI_PrintSessionData in_printSessionData);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID that identifies the print session; provided by GDAPI_QueuePrintSession.

in_printSessionData

This structure defines a given print session; See the “Print Session Data” section for details.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_CANT_CHANGE_MEDIA_SIZE,
  GDAPI_CANT_CHANGE_MEDIA_THICKNESS
  GDAPI_CANT_CHANGE_RESOLUTION,
  GDAPI_CANT_CHANGE_IMAGE_DIMENSION,
  GDAPI_CANT_CHANGE_BIT_DEPTH,
  GDAPI_CANT_CHANGE_INTERLEAVING
  See the “Print Session” section for details regarding when these values can be set.
- GDAPI_INVALID_ARGUMENT
  The data in in_printSessionData was not valid.
- GDAPI_FAILURE

5.28 GDAPI_CancelPrintSession

Description: GDAPI_CancelPrintSession is used to remove a print session; this can be used to cancel sessions that are queued or in progress.

After calling GDAPI_CancelPrintSession, GDAPI_EndPrintSession **must** be called to signal that the raster source has finished canceling the print session.

Syntax:

GDAPI_Status **GDAPI_CancelPrintSession**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId);

Parameters:

in_deviceId

The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID that identifies the print session; provided by GDAPI_QueuePrintSession.

Return Code:

- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_PRINT_SESSION_ID

5.29 GDAPI_ReadyForRelease

Description: GDAPI_ReadyForRelease is called to signal the DSI that the raster source has finished specifying data that has a scope of DataScope_PrintSession (see the "Data Scope" section for details regarding scope).

GDAPI_ReadyForRelease is called:

- After GDAPI_QueuePrintSession.
- After any, optional, calls to GDAPI_SetCurrentAttributeValue and GDAPI_UpdatePrintSessionData, either of which is used to specify data with a scope of DataScope_PrintSession.

The DSI will not send the RELEASE command from PrintSessionCallback until GDAPI_ReadyForRelease has been called – this means that the raster source should call GDAPI_ReadyForRelease as soon as possible to have the DSI start processing the print session.

Syntax:
GDAPI_Status **GDAPI_ReadyForRelease** (
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId);

Parameters:
in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession.

Return Code:

- GDAPI_SUCCESS

- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_FAILURE

5.30 GDAPI_StartPrintSession

Description: GDAPI_StartPrintSession signals the DSI that the raster source is ready to start the print session. The output device should perform any required initialization.

Before making this call the raster source must have received the RELEASE command from PrintSessionCallback.

The raster source will not start sending raster until PrintSessionCallback sends the SEND_RASTER command.

Syntax:

GDAPI_Status **GDAPI_StartPrintSession**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.

- GDAPI_FAILURE

5.31 GDAPI_StartImageTransfer

Description: GDAPI_StartImageTransfer is called by the raster source to signal the output device that the raster data for an image is pending. The output device should perform any required initialization related to imaging the raster data.

Images must be transferred to the device in the order they were defined in GDAPI_QueuePrintSession.

If this call fails the raster source must call GDAPI_EndImageTransfer to end the image and can optionally call GDAPI_EndPrintSession to end the print session.

Syntax:

GDAPI_Status **GDAPI_StartImageTransfer**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_FAILURE

5.32 GDAPI_SetImageTransferOptions

Introduced in GDAPI version: 2.0.1

Description: The options which can be set by this function effect how the following GDAPI functions operate:

- GDAPI_StartImageTransfer
- GDAPI_SendImageData
- GDAPI_EndImageTransfer

The options allow you to specify whether the functions listed above handle Control, Data, or both Control and Data; the reasons why this may be useful are outlined below.

- **Control** – the functions send control messages to the DSI, but do not deliver image data to the device.
- **Data** – the functions send image data to the device, but do not send any control messages to the DSI.
- **Control and Data** – *this is the default when the GDAPI DLL is loaded*, the "Start", "Send", and "End" functions send both control messages to the DSI and image data to the device.

The Control and Data options are selected using a bit mask; the bit masks available are:

- IMAGE_TRANSFER_OPTION_DATA
- IMAGE_TRANSFER_OPTION_CONTROL

The options can be changed at any time except the time between the calls to GDAPI_StartImageTransfer() and GDAPI_EndImageTransfer().

The following table summarizes how the arguments of GDAPI_SendImageData are used based on the image transfer options selected.

Argument	**Use of the argument when the *Control* option is selected**	**Use of the argument when the *Data* option is selected**
in_deviceId	Used to direct the call to the appropriate device.	Not used.
in_printSessionId	Used to direct the call to the appropriate print session.	Not used.
in_numBytesRaster	Can be used by the DSI to calculate the percentage complete for the current image.	Used to determine the length of the in_raster argument.
in_raster	Not used.	Used as the source of the image data to deliver to the device.

Setting the Control or Data options to effect the behavior of the GDAPI_StartImageTransfer, GDAPI_SendImageData, and GDAPI_EndImageTransfer functions can be useful for the following reasons.

Alternate Path of Delivery for Image Data

When there is a more direct "data path" from the Raster Source to the device than via the GDAPI, it is more efficient for the Raster Source to send the image data to the device via the direct path. Note that when the Raster Source delivers the data from the same platform as the location of the GDAPI DLL that delivering the data via the GDAPI should be the most efficient approach. In the case of an alternate path the Raster Source can set the image transfer options to Control and:

- Call GDAPI_StartImageTransfer to signal the DSI that the image is starting.
- Wait for the SEND_RASTER signal from the Print Session callback.
- Deliver the data via the alternate path.

  The Raster Source may also call GDAPI_SendImageData; when the Control option is set, this function will not deliver data, but can be used by the DSI to calculate a percentage complete figure for the current image. The DSI can calculate this based on the image size, pixel depth, and the number of bytes specified in the GDAPI_SendImageData call.
- Call GDAPI_EndImageTransfer to signal the DSI that the image is complete and that the media be ejected.

Image Data Delivered From a Separate Process

The raster source may want to deliver the image data from a process which is separate from the process which is handling control. To deliver data from a separate process, the Raster Source would have the two processes coordinate as follows:

"Control" Process	**"Data" Process**
Call GDAPI_SetImageTransferOptions to specify that this instance of the DLL handle only Control.	Call GDAPI_SetImageTransferOptions to specify that this instance of the DLL handle only Data.
The control process will handle all GDAPI functions, except delivery of data.	The control process will handle only the delivery of data.
Queue a print session and wait for the RELEASE signal from the Print Session callback.	
Call GDAPI_StartPrintSession	
Call GDAPI_StartImageTransfer to signal the DSI that the image is starting.	
	Call GDAPI_StartImageTransfer to open the "data port" to the device.
Wait for the SEND_RASTER signal from the Print Session callback.	
	Call GDAPI_SendImageData to deliver

	the image data to the device.
Optionally call GDAPI_SendImageData to allow the DSI to calculate a percentage complete figure for the current image.	
Call GDAPI_EndImageTransfer to signal the DSI that the image is complete and that the media be ejected.	
	Call GDAPI_EndImageTransfer to close the "data port" to the device.

Syntax:

bool **GDAPI_SetImageTransferOptions**(
unsigned short in_imageTransferOptionsFlag);

Parameters:

in_imageTransferOptionsFlag

This flag must be set to one of the following values:

```
IMAGE_TRANSFER_OPTION_DATA
IMAGE_TRANSFER_OPTION_CONTROL
IMAGE_TRANSFER_OPTION_DATA | IMAGE_TRANSFER_OPTION_CONTROL
```

Return Code:

- True if in_imageTransferOptionsFlag is set to a valid value; see the parameter description above.
- False if in_imageTransferOptionsFlag does not contain a valid value. Note that the value is not set if the function returns false.

5.33 GDAPI_GetImageTransferOptions

Introduced in GDAPI version: 2.0.2

Description: GDAPI_GetImageTransferOptions is called by the raster source to get the image transfer options; these will either be the default options, or the options as set by GDAPI_SetImageTransferOptions.

See GDAPI_SetImageTransferOptions for complete details of the use and effect of these options.

Syntax:

void **GDAPI_GetImageTransferOptions**(
unsigned short& out_imageTransferOptionsFlag);

Parameters:

out_imageTransferOptionsFlag

This will be set to one of the following values:

```
IMAGE_TRANSFER_OPTION_DATA
IMAGE_TRANSFER_OPTION_CONTROL
IMAGE_TRANSFER_OPTION_DATA | IMAGE_TRANSFER_OPTION_CONTROL
```

5.34 GDAPI_SendImageData

Description: GDAPI_SendImageData is called by the raster source to transfer the raster data to the output device. The GDAPI calls required for a given image are:

- GDAPI_StartImageTransfer
- The DSI signals it is ready for image data by sending the "SEND_RASTER" message via the PrintSessionCallback callback.
- GDAPI_SendImageData – called one or more times to transfer the image data.
- GDAPI_EndImageTransfer

If this call fails the raster source must call GDAPI_EndImageTransfer to end the image and can optionally call GDAPI_EndPrintSession to end the print session.

Note that the image data can be delivered to the imaging device without calling GDAPI_SendImageData. See GDAPI_SetImageTransferOptions for details. An alternate to delivering data via GDAPI_SendImageData is required in cases where:

- the data must pass through a hardware screening card and the card delivers data directly to the device.
- there is a more direct data path to the device than going through GDAPI_SendImageData, etc.

In these cases the protocol is as follows; note that calling "Start" and "End" are still required for the purposes of sending control signals to the device.

- GDAPI_StartImageTransfer
- The DSI signals it is ready for image data by sending the "SEND_RASTER" message via the PrintSessionCallback callback.
- Send image data via the *alternate* delivery path.
- GDAPI_EndImageTransfer

Image data can be very large; thus it is appropriate to mention some efficiency issues here:

- The amount of data sent for each call to GDAPI_SendImageData can effect throughput. For example, if the DSI is writing the data to a SCSI bus, maximum throughput may be achieved when raster is delivered in 64K byte blocks.

- It is the choice of the DSI whether it copies the raster data and returns when the copy is complete, or uses the raster data directly and returns when the raster data has been sent.
- Some output mechanisms used by the DSI may provide increased throughput when more than one output buffer is queued (overlapped output).
- GDAPI_SendImageData is a synchronous function call, I.E. it returns when the DSI is finished with the raster data. If a raster source must stay responsive to input requests, it should consider handling these calls in a separate thread.

Syntax:

GDAPI_Status **GDAPI_SendImageData**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId,
long in_numBytesRaster,
void* in_raster);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession.

in_numBytesRaster
The number of bytes of raster passed via the in_raster argument.

in_raster
A pointer to the stream of bytes which is the raster data.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_CLIPPED_IMAGE
  The image was larger than the media selected and the image has been clipped. Note that it is up to the DSI whether or not it supports clipping.

- GDAPI_IMAGE_TOO_LARGE
  This is an error – the print session should be canceled. The image is larger than the media selected.

- GDAPI_FAILURE

If a failure occurs, the raster source may have to cancel the current image by calling GDAPI_CancelCurrentImage.

5.35 GDAPI_EndImageTransfer

Description: GDAPI_EndImageTransfer is called by the raster source to signal the output device that an image has been completely transferred using GDAPI_SendImageData. For most devices, GDAPI_EndImageTransfer would signal the output device to begin the media handling required to unload the imaged media.

Syntax:

GDAPI_Status **GDAPI_EndImageTransfer**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession.

Return Code:

- GDAPI_SUCCESS

- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.

- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.

- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.

5.36 GDAPI_EndPrintSession

Description: GDAPI_EndPrintSession is called by the raster source when it has completed sending all of its images to the output device for a given print session.

The DSI will not release the next print session until GDAPI_EndPrintSession has been called; this also applies to print sessions which have been canceled by either the raster source or the DSI.

Calling GDAPI_EndPrintSession is also the signal to the DSI that the print session ID is now invalid.

Syntax:

GDAPI_Status **GDAPI_EndPrintSession**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session.
- GDAPI_WRONG_STATE_FOR_END_PRINT_SESSION
  One of the following has not been called before calling GDAPI_EndPrintSession: GDAPI_StartPrintSession or GDAPI_CancelPrintSession.

5.37 GDAPI_CancelCurrentImage

Description: GDAPI_CancelCurrentImage is called to signal that the raster output should be canceled for the current image and that the media be expelled. The output device should perform any required operations to cease imaging.

The raster source has two options following the call to GDAPI_CancelCurrentImage:

1. Wait for the PrintSessionCallback to issue the SEND_RASTER command for the next separation.
2. Call GDAPI_EndPrintSession to complete the print session.

Syntax:

GDAPI_Status **GDAPI_CancelCurrentImage**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
The ID to identify the session; supplied by GDAPI_QueuePrintSession.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NO_CURRENT_IMAGE
  There was no image currently being processed.
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.

5.38 GDAPI_GetLocalizedStringLength

Description: GDAPI_GetLocalizedStringLength is called to get the length of a locale specific string. All strings must be identified by a GDAPI_StringId.

Syntax:

GDAPI_Status **GDAPI_GetLocalizedStringLength**(
GDAPI_DeviceId in_deviceId,
GDAPI_LocaleId in_localeId,
GDAPI_StringId in_stringId,
int* out_length);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices. This can also be GDAPI_NO_VALUE – to allow this call before connecting to a device.

in_localeId
The ID of the locale to use for finding the string. If the locale is not supported then the length of the US English version of the string will be returned.

in_stringId
The ID of the string.

out_length

This specifies the length (number of characters) of the string associated with in_localeID and in_stringed; the count does **not** include the NULL terminating character.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNSUPPORTED_LOCALE
  The locale ID is not supported.
- GDAPI_UNKNOWN_STRING_ID
  The string ID is not supported.
- GDAPI_FAILURE

5.39 GDAPI_GetLocalizedString

Description: GDAPI_GetLocalizedString is called to get the locale specific value for a string. All strings must be identified by a GDAPI_StringId.

Syntax:

GDAPI_Status **GDAPI_GetLocalizedString**(
GDAPI_DeviceId in_deviceId,
GDAPI_LocaleId in_localeId,
GDAPI_StringId in_stringId,
int in_maxCharacters,
wchar_t* out_string);

Parameters:

in_deviceId

The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices. This can also be GDAPI_NO_VALUE – to allow this call before connecting to a device.

in_localeId

The ID of the locale to use for finding the string. If the locale is not supported then the US English version of the string will be returned.

in_stringId

The ID of the string to return.

in_maxCharacters

This specifies the maximum number of characters, including the NULL terminator, which can be held in the out_string buffer allocated by the raster source.

out_string

The buffer which will contain the locale specific string; note that this buffer must be large enough to hold the NULL terminated string.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNSUPPORTED_LOCALE
  The locale ID is not supported.
- GDAPI_UNKNOWN_STRING_ID
  The string ID is not supported.
- GDAPI_INSUFFICIENT_BUFFER
  The string could not fit into out_string and was not retrieved; use GDAPI_GetLocalizedStringLength to determine the string length.

5.40 GDAPI_SendMessage

Description: GDAPI_SendMessage is called if the raster source wants to send a message string to the output device; for example, the raster source could report a rasterization error. The DSI handles this message as it deems appropriate (if there is a UI, it may display it or it may just ignore the message).

Syntax:

GDAPI_Status **GDAPI_SendMessage**(
GDAPI_DeviceId in_deviceId,
GDAPI_PrintSessionId in_printSessionId,
const wchar_t* in_message);

Parameters:

in_deviceId

The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices.

in_printSessionId
: This can be used by a DSI to filter message based on print sessions; set this to GDAPI_NO_VALUE if the message is not associated with a print session.

in_message
: The buffer containing the NULL terminated message; this buffer is allocated by the Raster Source.

Return Code:

- GDAPI_SUCCESS
- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_UNKNOWN_PRINT_SESSION_ID
  in_printSessionId specified an unknown print session. Perhaps this function should not check the print session id… to allow a raster source to send a message for a print session which has been canceled by the DSI.
- GDAPI_FAILURE

5.41 GDAPI_GetMessageLength

Description: GDAPI_GetMessageLength is called when the raster source wants to determine the length of a message associated with a status code.

GDAPI_GetMessage is used to retrieve the actual message.

Syntax:

GDAPI_Status **GDAPI_GetMessageLength**(
GDAPI_DeviceId in_deviceId,
GDAPI_LocaleId in_localeId,
GDAPI_Status in_statusCode,
int* out_length);

Parameters:

in_deviceId
: The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices. This can also be GDAPI_NO_VALUE – to allow this call before connecting to a device.

in_localeId
: The locale to use for the message.

in_statusCode
The DSI will provide the message which is associated with this status code.

out_length
This specifies the length (number of characters) of the message associated with in_statusCode.

Return Code:

- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_INVALID_STATUS_CODE
  in_statusCode is not recognized or is in the wrong format.
- GDAPI_UNSUPPORTED_LOCALE
  in_localeId is not supported.

5.42 GDAPI_GetMessage

Description: GDAPI_GetMessage is called when the raster source wants to retrieve an error string for a status code returned by a GDAPI function.

Syntax:

GDAPI_Status **GDAPI_GetMessage**(
GDAPI_DeviceId in_deviceId,
GDAPI_LocaleId in_localeId,
GDAPI_Status in_statusCode,
int in_maxCharacters,
wchar_t* out_statusMessage);

Parameters:

in_deviceId
The ID that identifies the output device; this is one of the IDs returned by GDAPI_GetAvailableDevices. This can also be GDAPI_NO_VALUE – to allow this call before connecting to a device.

in_localeId
The locale to use for the message.

in_statusCode
The DSI will provide the message which is associated with this status code.

in_maxCharacters
This specifies the maximum number of characters, including the NULL

terminator, which can be held in the out_statusMessage buffer allocated by the raster source.

out_statusMessage
The buffer containing the NULL terminated message; this buffer is allocated by the Raster Source and can hold (in_numCharacters-1) characters + the NULL character. If the message is longer it will be truncated.

Return Code:

- GDAPI_NOT_CONNECTED_TO_DEVICE
  GDAPI_DeviceConnect has not been called.
- GDAPI_UNKNOWN_DEVICE_ID
  in_deviceId specified an unknown device.
- GDAPI_INSUFFICIENT_BUFFER
  The message could not fit into out_statusMessage and was not returned; GDAPI_GetMessageLength can be used to retrieve the length of a message.
- GDAPI_INVALID_STATUS_CODE
  in_statusCode is not recognized or is in the wrong format.
- GDAPI_UNSUPPORTED_LOCALE
  in_localeId is not supported.

Notes:

The raster source must allocate a buffer of suitable length to pass to the output device. The output device should populate the error string with the appropriate information and terminate it with a NULL character. Memory management is the responsibility of the raster source.

6. ERROR HANDLING

If an error occurs the raster source should issue a report through its reporting mechanisms. The report should consist of the function return code plus a textual representation of the message. The error string is obtained using the GDAPI_GetLocalizedString function call.

It is also important to note that if the output device reports an error during imaging (GDAPI_SendImageData for example) that the raster source must complete the output cycle. This includes calling GDAPI_EndImageTransfer and GDAPI_EndPrintSession as required.

7. ATTRIBUTES

7.1 Core Attribute Summary

This section lists the "core attributes" which must be supported by every device in order to support the GDAPI.

Attribute	Description
DEVICE_ATTRIBUTE	**This is a special attribute used to get the list of attributes supported by a device**. It returns both core attributes and non-core attributes. Core attributes are returned to allow the raster source to treat all attributes in a consistent manner.
QUEUE_LENGTH	Used to get queue length.
RESOLUTION	Used to get the resolutions supported.
DEVICE_MEDIA	Used to get the device media supported.
MEDIA_SIZE	Used to get the size information for device media of discrete sizes and device media which is roll fed.
BIT_DEPTH	Used to determine the bit depths supported by a device. For example, a half tone device may only support a bit depth of one.
INTERLEAVING	Used to determine the interleaving options supported by a DSI for a given device.

The details of these attributes are described in the following sections.

7.2 Non-Core Attribute Summary

This section describes some common non-core attributes; attributes which are supported or not depending on the capabilities of the imaging device – they are not required by other parts of the GDAPI.

Attribute	Description
MEDIA_AVAILABLE	Used to get the supported media which are currently available.
COLOR_NAME	Used to get the device color names and/or the color order for a DEVICE_MEDIA which has separations that require specific color name for separations and/or a specific color order for separations.

INVERT_RASTER	Used to specify whether or not the DSI will invert the raster data delivered by the Raster Source.

The details of these attributes are described in the following sections.

7.3 Data Scope

7.3.1 Overview

Data scope is a characteristic of attributes and is reported for each attribute when reading the DEVICE_ATTRIBUTE values, which describe the attributes supported by a given device. The concept of data scope can also apply to print session data values specified via GDAPI_QueuePrintSession and GDAPI_UpdatePrintSessionData.

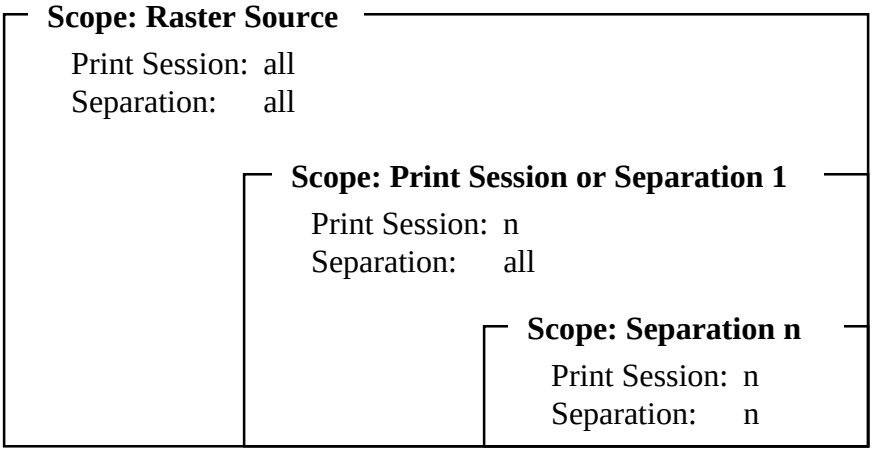
Data scope is applicable only for attributes that can be written, I.E. for an attribute that the raster source can **set a value** by calling GDAPI_SetCurrentAttributeValue and specifying a print session id (or a wild card), a separation id (or a wild card) and a value. Attributes that are read only have a data scope of DataScope_NotApplicable. Data Scope can be one of:

- **DataScope_NotApplicable** – for attributes that are read-only.
- **DataScope_PrintSession** – the attribute value must be specified before the call to GDAPI_ReadyForRelease.
- **DataScope_Separation_1** – the attribute value must be specified before the 1st call to GDAPI_StartImageTransfer.
- **DataScope_Separation_n** – the attribute value must be specified before the call to GDAPI_StartImageTransfer, for the separation to which the value applies.

Data scope also determines whether a attribute *value* set by the raster source can be specific to a print session, or to a separation. Only attributes with a scope of DataScope_Separation_n can be specific to a separation.

The following diagram shows how an attribute value can be set for different scopes – to apply to:

- All print sessions and separations generated by a raster source,
- all separations in a given print session or
- a specific separation in a print session.



7.3.2 Examples

The following examples illustrate some of the motivation for data scope.

Note that the concept of data scope can be applied to all print session data, not just attributes. For these examples, print session data that is not set via attributes are used in each example – to avoid describing device specific attributes.

- **DataScope_PrintSession**
  An example of print session data that could have this scope is Media Size. This information must be supplied before the call to GDAPI_ReadyForRelease. The reason for such a restriction is to keep the device busy – to supply the device with all information required to load the media for a given print session. The alternative – to have the device wait for such information – would slow down the processing of print sessions.

- **DataScope_Separation_1**
  An example of print session data that has this scope is Image Dimensions. This information is not required by a device to load media, but because it applies to *all* separations in a print session, it must be specified before the 1st separation is processed. Providing this scope allows the raster source more time to "discover" this information when ripping.

- **DataScope_Separation_n**
  An example of print session data that could have this scope is Image Dimensions. This information is not required by a device to load media, and *could* change from one separation to another as long as each image still fit the Media Size selected.

7.3.3 Data Scope Errors

Data scope effects error checking as follows:

- Setting a value with **DataScope_PrintSession** cannot be done after the call to GDAPI_ReadyForRelease and will return: GDAPI_OUT_OF_SCOPE.

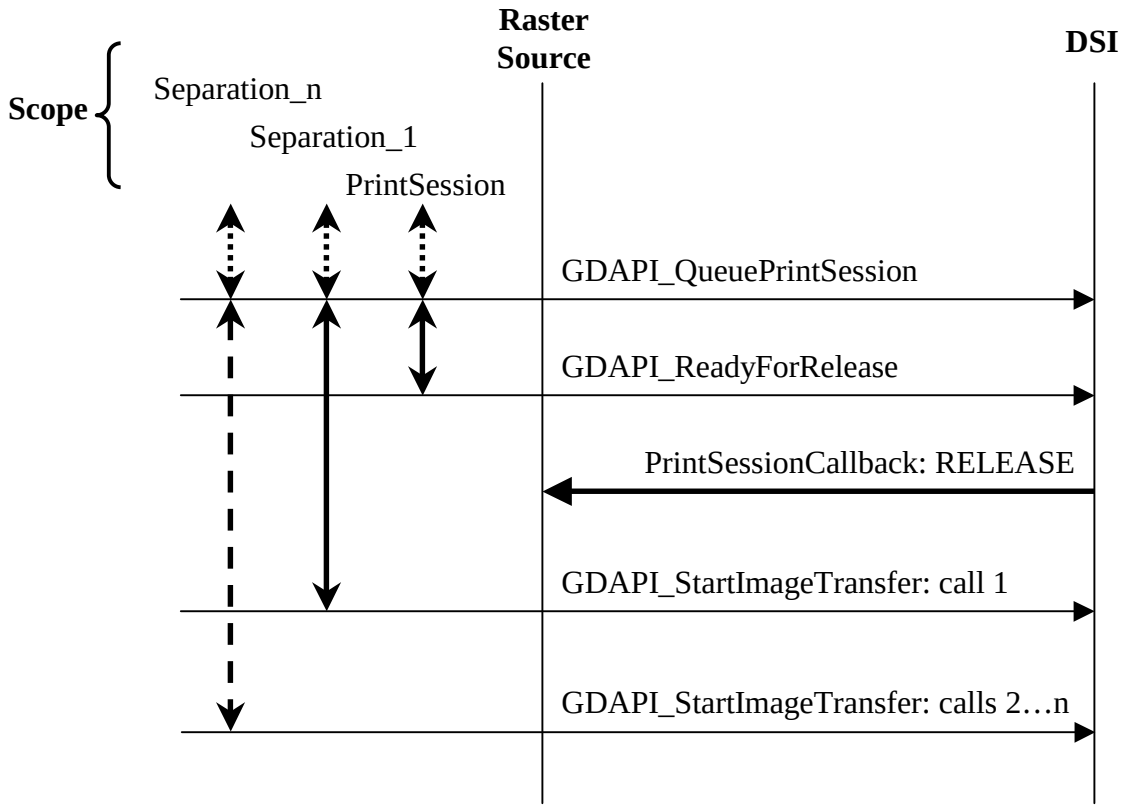
- Setting a value with **DataScope_Separation_1** cannot be done after the *first* call to GDAPI_StartImageTransfer and will return: GDAPI_OUT_OF_SCOPE.
- Setting a value with **DataScope_Separation_n** cannot be done after the call to GDAPI_StartImageTransfer and will return: GDAPI_OUT_OF_SCOPE.

7.3.4 Summary

The following table and diagram summarize when an attribute can first be set and when it can last be set – depending upon the data scope of the attribute. Note that if an attribute is set *before* the call to GDAPI_QueuePrintSession, which provides a print session ID, then the raster source does not know the print session ID and can only set the attribute value which applies to the raster source – I.E. not specific to a print session or separation.

The *value* for an attribute with a data scope of Separation_n can be “discovered” by the raster source and then specified at any point up until processing the separation to which it applies.

Key	Value Can be Specific to		
	Raster Source	Print Session	Separation
.....	✓		
———	✓	✓	
– – – –	✓	✓	✓



Note that this diagram shows only the function calls which apply to the “cut off” points for setting attribute values in various circumstances – other function calls required to process a print session have been left out for clarity.

Whenever an attribute value is set, it will effect the next separation that it applies to.

The arguments of a GDAPI_SetCurrentAttributeValue call, example shown below, which effect attribute scope are “printSessionId” and “separationId”:

```
status = GDAPI_SetCurrentAttributeValue(
    deviceId,
    printSessionId,
    separationId,
    L’SOME_ATTRIBUTE’
    Sizeof( long ),
    (void*) &variableOfTypeLong );
```

The following table shows which combinations settings for “printSessionId” and “separationId” are legal for attributes of various scopes.

Scope	printSessionId	separationId	Comment
Any Scope	GDAPI_NO_VALUE	m	**Illegal**
PrintSession, Separation_1	GDAPI_NO_VALUE	GDAPI_NO_VALUE	Legal
	n	GDAPI_NO_VALUE	Legal
Separation_n	GDAPI_NO_VALUE	GDAPI_NO_VALUE	Legal
	n	GDAPI_NO_VALUE	Legal
	n	m	Legal

7.4 Default Values

This section describes the use of default attribute values and provides an example of their use.

Default values for attributes serve the following purposes:

- Describe the value that is used when the raster source does not specify any value for an attribute.
- Allow a user interface presenting a list of choices for an attribute to show the value that will ‘be in effect’ by default.

Example:

The ‘invert raster’ attribute allows a Raster Source to specify whether or not the DSI will invert the raster on an image:

- GDAPI_GetAvailableAttributeValue returns ‘true’ and ‘false’
- GDAPI_GetDefaultAttributeValue returns ‘false’.
  This means that, by default, the DSI will not invert the raster. The Raster Source only need call GDAPI_SetCurrentAttributeValue if the user wants the raster inverted.

7.5 Core Attribute Details

7.5.1 DEVICE_ATTRIBUTE

Note:
This attribute is used to retrieve the list of all attributes supported by a device and supported by a given DSI.

DEVICE_ATTRIBUTE is the attribute name used retrieve the list of all device attributes, both core and non-core.

Note: When a Raster Source receives an "attribute change notification" for the DEVICE_ATTRIBUTE attribute, the Raster Source should take the following steps:

- Read the list of values for the DEVICE_ATTRIBUTE and compare this list to the original list read at startup.
- If the DEVICE_ATTRIBUTE reports a new attribute, read the values for the new attribute.
- If the DEVICE_ATTRIBUTE drops an attribute, drop the attribute from the Raster Source.

The information returned for each attribute includes:

Device Attribute: Field and Comment
attributeName This name is used as a key to this attribute when calling functions to get or set attribute values. The name is passed back by the AttributeChangedCallback function to indicate that the DSI has changed the available values for this attribute.
AttributeNameStringId This string ID can be passed to the GDAPI_GetLocalizedString function to retrieve a localized version of the attribute name.
AttributeType The type of the attribute. This allows for the identification of standard attribute types which can be handled by the raster source in a generic way. The main goal of the attribute type is to allow a raster source to handle standard attributes in a generic way, and to ease the burden of dealing with new attributes. AttributeType_Custom indicates a structure which does not fit into a standard structure. The attribute structure is described here or by the DSI. See the “Standard Attribute Types” section for the definition of the data structures which correspond to the standard types and for examples of how various standard GDAPI_AttributeType types are handled.
DataScope This field identifies the possible scope of the attribute. See the “Data Scope” section for a description of how attributes of different scopes are used.
RasterSourceAccess The raster source access allowed for the attribute: read, write, or read and write.
SelectionModel This field identifies whether or not the attribute presents its data as a list from which a selection is made. If the attribute is a type of a list, this field says whether the list supports single or multiple selection.

Data:

```
typedef struct
```

```
{
    wchar_t              attributeName[GDAPI_SHORT_STRING];
    GDAPI_StringId       attributeNameStringId;
    GDAPI_AttributeType  attributeType;
    GDAPI_DataScope      dataScope;
    GDAPI_Access         rasterSourceAccess;
    GDAPI_SelectionModel selectionModel;

} GDAPI_AttributeInformation;
```

Related Data:

```
typedef enum
{
    AttributeType_Custom      = IDS_ATTR_TYPE_CUSTOM,

    AttributeType_String      = IDS_ATTR_TYPE_STRING,
    AttributeType_StringList  = IDS_ATTR_TYPE_STRING_LIST,

    AttributeType_Long        = IDS_ATTR_TYPE_LONG,
    AttributeType_LongList    = IDS_ATTR_TYPE_LONG_LIST,

    AttributeType_Double      = IDS_ATTR_TYPE_DOUBLE,
    AttributeType_DoubleList  = IDS_ATTR_TYPE_DOUBLE_LIST,

    AttributeType_Bool        = IDS_ATTR_TYPE_BOOL,
    AttributeType_RangeLong   = IDS_ATTR_TYPE_RANGE_LONG,
    AttributeType_RangeDouble = IDS_ATTR_TYPE_RANGE_DOUBLE

} GDAPI_AttributeType;

typedef enum
{
    DataScope_NotApplicable = IDS_DATA_SCOPE_NOT_APPLICABLE,
    DataScope_PrintSession  = IDS_DATA_SCOPE_PRINT_SESSION,
    DataScope_Separation_1  = IDS_DATA_SCOPE_SEPARATION_1,
    DataScope_Separation_n  = IDS_DATA_SCOPE_SEPARATION_N

} GDAPI_DataScope;

typedef enum
{
    Access_Read      = IDS_ACCESS_READ,
    Access_Write     = IDS_ACCESS_WRITE,
    Access_ReadWrite = IDS_ACCESS_READ_WRITE

} GDAPI_Access;

typedef enum
{
    SelectionModel_NotApplicable =
        IDS_SELECTION_MODEL_NOT_APPLICABLE,

    SelectionModel_Single        = IDS_SELECTION_MODEL_SINGLE,
    SelectionModel_Multiple      = IDS_SELECTION_MODEL_MULTIPLE

} GDAPI_SelectionModel;
```

Here is an example of how to retrieve the list of attributes supported by a DSI; note that each DSI must also document the attributes it supports:

```
GDAPI_AttributeInformation attribute;
GDAPI_Status status = 0;
bool done = false;
GDAPI_DeviceId device = 1;
int numBytes = sizeof( GDAPI_AttributeInformation );
int index = 0;

while( ! done )
{
    status = GDAPI_GetAvailableAttributeValue(
        device,
        L“DEVICE_ATTRIBUTE”,
        index,
        numBytes,
        (void *) (&attribute) );

    if( ! GDAPI_SUCCEEDED( status ) )
    {
        // report the error
        // take an appropriate action
        done = true;
    }
    else if( status == GDAPI_END_OF_LIST )
    {
        done = true;
    }
    else
    {
        // allocate space for another “element”
        // store the element

        index++;
    }

} // while
```

OR

```
int attributeCount;

status = GDAPI_GetNumAvailableAttributeValues(
    device,
    L“DEVICE_ATTRIBUTE”,
    &attributeCount );

// Allocate space for all attributes based on the value in
// attributeCount and then retrieve each attribute value as
// shown above.
```

7.5.2 QUEUE_LENGTH

The QUEUE_LENGTH attribute is used to determine the queue length of a device. In the case of a successful call the queue length will be 1 or more.

DEVICE_ATTRIBUTE values for QUEUE_LENGTH:

Field	Value
Name	QUEUE_LENGTH
Type	AttributeType_Long
Data Scope	DataScope_NotApplicable

Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

The queue length signifies the maximum number of sessions that can exist in the DSI queue.

If the raster source cannot get the queue length, the device may still be capable of imaging; the raster source will simply not know the maximum number of print sessions it can queue.

Example:

```
long queueLength;
GDAPI_Status status = 0;
GDAPI_DeviceId device = 1;
int index = 0;

status = GDAPI_GetAvailableAttributeValue (
    device,
    L“QUEUE_LENGTH”,
    index,
    sizeof( long ),
    (void *) (&queueLength) );
```

7.5.3 RESOLUTION

The RESOLUTION attribute is used to retrieve the resolutions supported by the output device.

The raster source is required to retrieve RESOLUTION data as this data is required for defining print sessions; see GDAPI_QueuePrintSession and GDAPI_UpdatePrintSessionData.

DEVICE_ATTRIBUTE values for RESOLUTION:

Field	**Value**
Name	RESOLUTION
Type	AttributeType_Custom
Data Scope	DataScope_NotApplicable
Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

Data:

```
typedef struct
{
    double                 FastScan;
    double                 SlowScan;
    GDAPI_ResolutionUnits  Units;

} GDAPI_Resolution;

typedef enum
{
    ResolutionUnits_DotsPerInch         = IDS_RESOLUTION_DPI,
```

```
        ResolutionUnits_DotsPerCentimeter   = IDS_RESOLUTION_DPCM,
        ResolutionUnits_SpotSizeMicrometers = IDS_RESOLUTION_UNITS_SSM

    } GDAPI_ResolutionUnits;
```

Note that “Spot Size Micrometers” is the resolution expressed as the size of the spot in micrometers, I.E. a spot size of 10 micrometers would allow 100 spots / millimeter or 100 spots / centimeter.

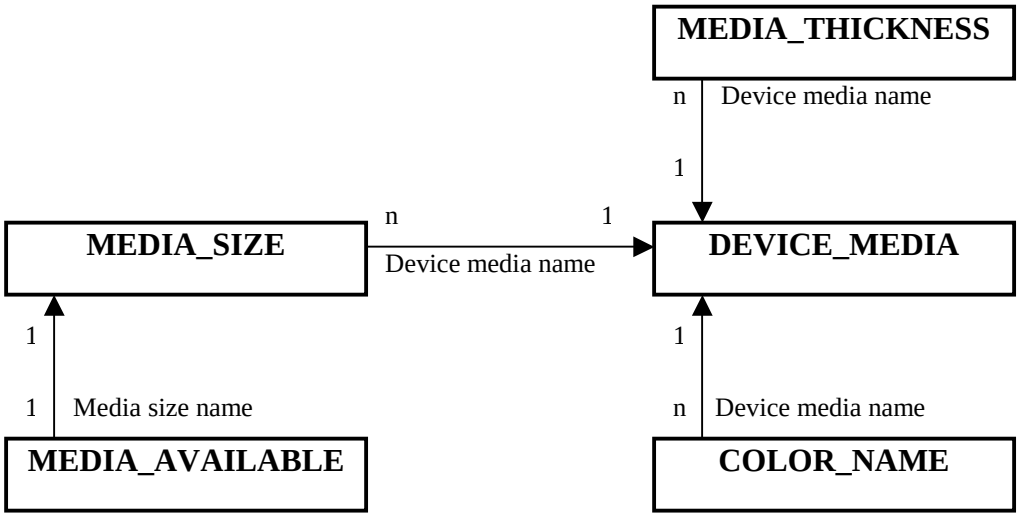
Example:

See previous attributes for an example.

7.5.4 Relationship Between Media Attributes

There are several media attributes: DEVICE_MEDIA, MEDIA_SIZE, MEDIA_AVIALABLE, MEDIA_THICKNESS and COLOR_NAME. The following diagram shows the relationship between these attributes. The attributes are described in the sections which follow.

- The arrows show which attribute refers to another. For example, COLOR_NAME refers to DEVICE_MEDIA using the “Device media name”.
- The numbers on the arrows show the cardinality of the relationships. Note that ‘n’ can be 0 in all cases.



7.5.5 DEVICE_MEDIA

The DEVICE_MEDIA attribute is used to retrieve the types of device media supported by an output device; this provides more information about a given device media than the MEDIA_SIZE attribute, but is not required for other GDAPI functions.

DEVICE_ATTRIBUTE values for DEVICE_MEDIA:

Field	Value
Name	DEVICE_MEDIA
Type	AttributeType_Custom
Data Scope	DataScope_NotApplicable
Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

Data:

```
typedef struct
{
    wchar_t             DeviceMediaName[GDAPI_SHORT_STRING];
    wchar_t             Manufacturer[GDAPI_SHORT_STRING];
    GDAPI_MediaHandling MediaHandling;
    GDAPI_MediaMaterial MediaMaterial;
    bool                Positive;
    bool                RasterInvertedByDsi;
} GDAPI_DeviceMedia;
```

Related Data:

```
typedef enum
{
    MediaHandling_SingleSided   =
        IDS_MEDIA_HANDLING_SINGLE_SIDED,

    MediaHandling_FlipLeftRight =
        IDS_MEDIA_HANDLING_FLIP_LEFT_RIGHT,

    MediaHandling_FlipTopBottom =
        IDS_MEDIA_HANDLING_FLIP_TOP_BOTTOM

} GDAPI_MediaHandling;

typedef enum
{
    MediaMaterial_Unknown          = IDS_MEDIA_MATERIAL_UNKNOWN,

    MediaMaterial_AluminumPlate    =
        IDS_MEDIA_MATERIAL_ALUMINUM_PLATE,

    MediaMaterial_PolyesterPlate   =
        IDS_MEDIA_MATERIAL_POLYESTER_PLATE,

    MediaMaterial_FlexographyPlate =
        IDS_MEDIA_MATERIAL_FLEXO_PLATE,

    MediaMaterial_Film             = IDS_MEDIA_MATERIAL_FILM,

    MediaMaterial_HalftoneProof    =
        IDS_MEDIA_MATERIAL_HALFTONE_PROOF,

    MediaMaterial_BluelineProof    =
        IDS_MEDIA_MATERIAL_BLUELINE_PROOF

} GDAPI_MediaMaterial;
```

Field	Description

DeviceMediaName	The name of this media – use this name to specify a media in QueuePrintSession.
Manufacturer	The name of the media manufacturer.
MediaHandling	Double sided media can dictate how it is handled after the first side has been printed, this field specifies the handling after the first side has been printed.
MediaMaterial	The basic material from which the media is made; see GDAPI_MediaMaterial. The purpose of identifying the media material is so that the imaging device can change any settings when dealing with different materials.
Positive	This flag is true for a positive media and false for a negative media. The GDAPI defines a positive media and a negative media as follows: • Positive Media – a pixel value of 0 in the *final* image data (I.E *after* any data inversion) is a transparent pixel on the media. • Negative Media – a pixel value of 0 in the *final* image data (I.E *after* any data inversion) is a colored pixel on the media. This flag has two implications for raster data: • **The size of the image required.** For a negative media, all non-exposed areas are "colored"; this means that in order to create transparent areas, for the background of an image, that the image must be the size of the plate image area (the area inside the margins). • **Whether or not to invert the raster data** Typically the raster data must be inverted for negative media; however, check the value of the RasterInvertedByDsi field – if this is set to true, then the raster data does not need to be inverted to obtain a positive image.
RasterInvertedByDsi	This flag is true if the raster data is inverted by the DSI. Note that a DSI only has to support the invert operation for half-tone data – con-tone data should be inverted by the raster source if a negative image is required as the inversion of con-tone data may also require color adjustments.

Notes:

- Each device media must have a unique name.
- Some proofing media requires that color separations are applied in a specific order to obtain the correct result, or that specific color names be used (to allow the

device to set up for the color). For such media the color name information is obtained using the COLOR_NAME attribute. Material_HalftoneProof is such a media type.

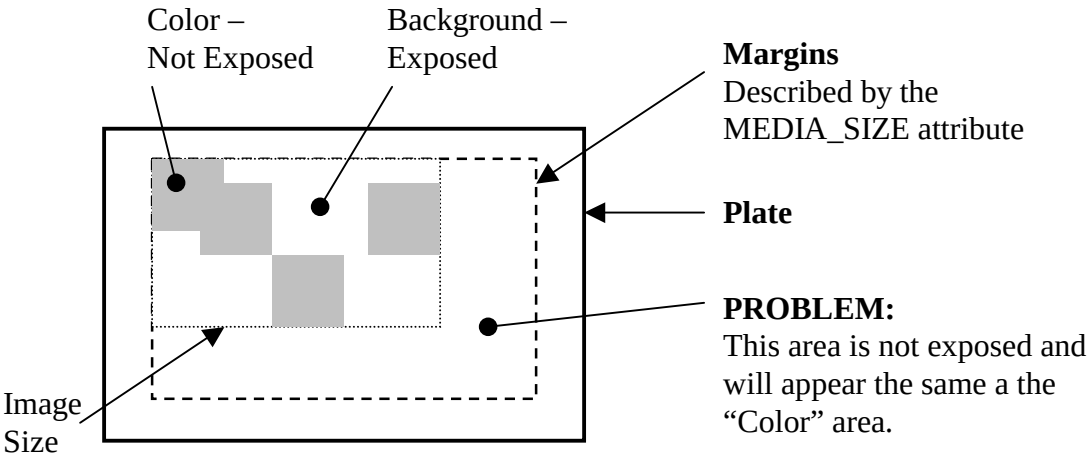
- See the following sub-section for the implications of Negative Media.

7.5.6 Implications of “Negative Media”

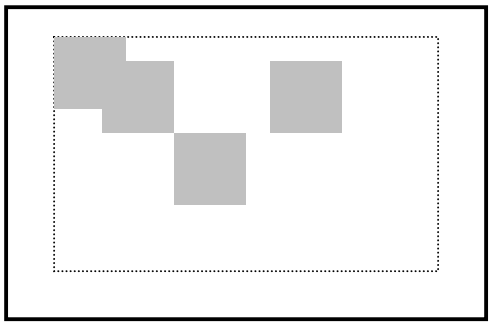
Note:
The GDAPI definition of negative media is: a pixel value of 0 in the *final* image data (I.E *after* any data inversion performed by the DSI) is a colored pixel on the media.

This section outlines the implications of negative media – which can effect the size of image data and how image offsets are handled.

The following diagram shows the result of applying an image to a negative media:



To avoid the problem shown in the diagram above, the image size must be increased to fit the area on the plate which can be imaged (defined by the margins provided by



the MEDIA_SIZE attribute).

This also means that image shifts for negative media must be handled by making the

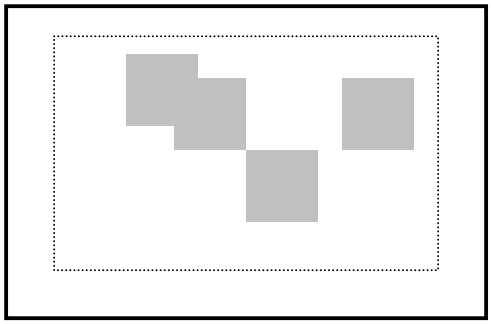


image the same size as the area which can be imaged and shifting the “non-background” data within the image.

An image shift for positive media can be more easily handled by the SHIFT_IMAGE attribute – if this attribute is supported by the DSI.

7.5.7 MEDIA_SIZE

The MEDIA_SIZE attribute is used to retrieve the sizes of device media supported by an output device.

The raster source is required to retrieve MEDIA_SIZE data as mediaSizeName is required to define a print session; see GDAPI_QueuePrintSession and GDAPI_UpdatePrintSessionData.

DEVICE_ATTRIBUTE values for MEDIA_SIZE:

Field	Value
Name	MEDIA_SIZE
Type	AttributeType_Custom
Data Scope	DataScope_NotApplicable
Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

MEDIA_SIZE links a DEVICE_MEDIA to a given size and may describe media of discrete sizes or media which is variable in size; see the description of the “dimensions” field for details.

Call GDAPI_GetAvailableAttributeValue once to retrieve each MEDIA_SIZE structure.

Data:

```
typedef struct
{
    wchar_t                 deviceMediaName[GDAPI_SHORT_STRING];

    wchar_t                 mediaSizeName[GDAPI_SHORT_STRING];
    GDAPI_MediaDimensions   dimensions;
    GDAPI_MediaMargins      margins;
```

```
    GDAPI_MediaThickness   MediaThickness;

} GDAPI_MediaSize;
```

Related data:

```
typedef struct
{
    double               MinWidth;
    double               MaxWidth;
    double               MinLength;
    double               MaxLength;
    GDAPI_UnitOfMeasure  Units;

} GDAPI_MediaDimensions;

typedef struct
{
    double               HomeEdge;
    double               AwayEdge;
    double               LeadingEdge;
    double               TrailingEdge;
    GDAPI_UnitOfMeasure  Units;

} GDAPI_MediaMargins;

typedef struct
{
    double thickness;
    GDAPI_UnitOfMeasure unitOfMeasure;

} GDAPI_MediaThickness;

typedef enum
{
    UnitOfMeasure_Inches      = IDS_UNITS_INCHES,
    UnitOfMeasure_Centimeters = IDS_UNITS_CM,
    UnitOfMeasure_Millimeters = IDS_UNITS_MM,
    UnitOfMeasure_MicroMeters = IDS_UNITS_MICRO_METERS,
    UnitOfMeasure_Mills       = IDS_UNITS_MILLS,
    UnitOfMeasure_Points      = IDS_UNITS_POINTS

} GDAPI_UnitOfMeasure;
```

The following table describes the fields in the GDAPI_MediaSize record.

Field	Description
deviceMediaName	This is the name of one of the device media returned for the DEVICE_MEDIA attribute and associates the MEDIA_SIZE structure with DEVICE_MEDIA structure.
mediaSizeName	The name of the media size; this name is synonymous with a device media of a discrete size or a device media which is roll fed. This name is used to specify a device media (of a given size or device media which is roll fed) for a print session.
dimensions	The physical dimensions of the media.

	Cut Sheet Media (discrete sizes) MinWidth = MaxWidth, MinLength = MaxLength. **Roll Fed Media** MinWidth = MaxWidth, MinLength < MaxLength. **Variable in Both Dimensions** MinWidth < MaxWidth, MinLength < MaxLength.
margins	The margins describe the amount of "non imaging space" which is required at the edges of the media. This can be dictated by how the media is held by the output device (clamp, vacuum) and by the media itself. The margins describe the amount of space from each edge that can not be imaged. It is device dependent whether or not an image shifted outside the margins is clipped. The margins have the following definitions: • HomeEdge – the edge where the slow scan begins. • AwayEdge – the edge where the slow scan ends. • LeadingEdge – the edge where the fast scan begins. • TrailingEdge – the edge where the fast scan ends. Typically the HomeEdge and the AwayEdge are at either end of the imaging drum and the LeadingEdge and TrailingEdge are where the media clamps into the imaging drum.
MediaThickness	The thickness field of the GDAPI_MediaThickness will either be GDAPI_VARIABLE_THICKNESS or it will be a given thickness. If the thickness is variable there are some implications for queuing a print session: • To select a specific Media Size when queuing the print session, you must specify the Media Size (which has a variable thickness) and a thickness value using the MediaThickness field. • Valid thickness values for various Device Media are listed by the MEDIA_THICKNESS attribute.

Notes:

- Rectangular media of discrete sizes will have two media size names, one for landscape mode and a second for portrait mode.
- The media returned are not guaranteed to be currently available to the output device; a cassette or bin may need to be loaded before the output device is capable

of imaging them. The MEDIA_AVAILABLE attribute can be used to determine which media are currently loaded in the device's cassettes.

7.5.8 MEDIA_THICKNESS

The MEDIA_THICKNESS attribute is used to get the valid thickness values for a given DEVICE_MEDIA. The thickness value obtained from this attribute can be specified in the call to GDAPI_QueuePrintSession or GDAPI_UpdatePrintSession data, but is not required unless the MEDIA_SIZE attribute reports a media size with a thickness value of GDAPI_VARIABLE_THICKNESS. In the case a media size with a variable thickness, the print session must specify both the media size and the thickness.

The following table shows the relationship between the MEDIA_SIZE thickness field and the Print Session Data MediaThickness field:

	MEDIA_SIZE thickness field = **GDAPI_VARIABLE_THICKNESS**	MEDIA_SIZE thickness field = 'n', where 'n' is a **discrete thickness**
Print Session Data MediaThickness field **Specified**	Print Session Data thickness is used	Media Size thickness is used
Print Session Data MediaThickness field **Not Specified**	Error, thickness must be specified.	Media Size thickness is used

DEVICE_ATTRIBUTE values for MEDIA_THICKNESS:

Field	**Value**
Name	MEDIA_THICKNESS
Type	AttributeType_Custom
Data Scope	DataScope_NotApplicable
Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

Call GDAPI_GetAvailableAttributeValue to determine the media thickness for a each device media. The following structure is used to retrieve media thickness information:

```
typedef struct
{
    wchar_t deviceMediaName[ GDAPI_SHORT_STRING ];
    double thickness;
    GDAPI_UnitOfMeasure unitOfMeasure;

} GDAPI_DeviceMediaThickness;
```

- deviceMediaName
  This is the name of the media returned for the DEVICE_MEDIA attribute.

- thickness
  One thickness is returned for each call to GDAPI_GetAvailableAttributeValue.

- unitOfMeasure
  The unit measure for the media thickness.

There is also a related data structure used to specify the media thickness for a print session. This structure does not need to contain the device media name as the device media is associated with a print session via the MEDIA_SIZE.

```
typedef struct
{
    double thickness;
    GDAPI_UnitOfMeasure unitOfMeasure;

} GDAPI_MediaThickness;
```

7.5.9 BIT_DEPTH

The BIT_DEPTH attribute is used to determine the bit depths supported by a device. For example: a half-tone device would support only a bit depth of 1, while a proofing device would support a bit depth of greater than one – where each pixel of data contained more than one bit of color information.

The raster source is required to retrieve BIT_DEPTH data as bit depth is required to define a print session; see GDAPI_QueuePrintSession and GDAPI_UpdatePrintSessionData.

DEVICE_ATTRIBUTE values for BIT_DEPTH:

Field	**Value**
Name	BIT_DEPTH
Type	AttributeType_Long
Data Scope	DataScope_NotApplicable
Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

Call GDAPI_GetAvailableAttributeValue to determine bit depth options for a given device.

7.5.10 INTERLEAVING

This attribute is used to determine the interleaving options supported by a DSI for a given device.

Interleaving indicates how the **raster source** must combine, or separate, the raster data to form the final images in a print session.

The raster source is required to retrieve INTERLEAVING data as it is required to define a print session; see GDAPI_QueuePrintSession and GDAPI_UpdatePrintSessionData.

DEVICE_ATTRIBUTE values for INTERLEAVING:

Field	Value
Name	INTERLEAVING
Type	AttributeType_Long
Data Scope	DataScope_NotApplicable
Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

Call GDAPI_GetAvailableAttributeValue to determine interleaving options supported by a DSI for a given device.

Note that the values returned for INTERLEAVING must be cast to an GDAPI_Interleaving enumeration type:

```
typedef enum
{
    Interleaving_Separation = IDS_INTERLEAVING_SEPARATION,
    Interleaving_Bit        = IDS_INTERLEAVING_BIT,
    Interleaving_Scanline   = IDS_INTERLEAVING_SCANLINE,
    Interleaving_Band       = IDS_INTERLEAVING_BAND

} GDAPI_Interleaving;
```

Interleaving Option	Description
Interleaving_Separation	This option, if supported, would be used by the raster source to produce half-tone images – images with a bit depth of one. The raster source would place each color in its own separation before submitting the separations in a print session.
Interleaving_Bit	This option, if supported, would be used by the raster source to produce an image with a bit depth greater than one. The raster source would build each pixel in an image by combining the bits from each separation.
Interleaving_Scanline	This option, if supported, would be used by the raster source to produce an image with a bit depth greater than one. The raster source would build each scan line in an image by combining the scan lines from each separation.
Interleaving_Band	This option, if supported, would be used by the raster source to produce an image with a bit depth greater than one. The raster source would build each band in an image by combining the bands from each separation. The combined band would

	then be sent with one GDAPI_SendImageData command.

7.6 Non-Core Attribute Details

7.6.1 MEDIA_AVAILABLE

The MEDIA_AVAILABLE attribute is used to determine which media sizes are currently loaded on the device: discrete media or roll fed media. If the device does not have cassettes for holding media, then nothing will be returned for the MEDIA_AVAILABLE attribute.

One value is returned for each cassette **loaded**. Note that different cassettes may hold the same media.

The AttributeChangedCallback will be called for this attribute when:

- If ‘n’ represents the number of plates in the cassette:
  - When ‘n’ changes from n = 0 to n > 0.
  - When ‘n’ changes from n > 0 to n = 0.
- A cassette is loaded with a different media.

DEVICE_ATTRIBUTE values for MEDIA_AVAILABLE:

Field	**Value**
Name	MEDIA_AVAILABLE
Type	AttributeType_Custom
Data Scope	DataScope_NotApplicable
Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

Data:

```
typedef struct
{
    wchar_t  MediaSizeName[GDAPI_SHORT_STRING];
    int      Cassette;

} GDAPI_MediaAvailable;
```

Field	**Description**
MediaSizeName	This is the name of one of the media sizes returned for the MEDIA_SIZE attribute.
Cassette	The cassette used for a media size. This is information which can be displayed or ignored.

7.6.2 COLOR_NAME

The COLOR_NAME attribute is used to get:

- The order in which color separations must be applied for certain DEVICE_MEDIA;
- The "device color" names for colored media which effect settings on the imaging device. The device color is the name of a color as it is understood by the device – I.E. when a device color is specified, the device may adjust itself to handle this specific color.

Color Order
The COLOR_NAME attribute returns values in the order in which the separations should be defined by GDAPI_QueuePrintSession for this media. This is applicable for any proofing material where changing the order of the color separations can change the result on the proof, I.E. C, M, Y, K color balance.

DEVICE_ATTRIBUTE values for COLOR_NAME:

Field	Value
Name	COLOR_NAME
Type	AttributeType_Custom
Data Scope	DataScope_NotApplicable
Raster Source Access	Access_Read
Selection Model	SelectionModel_NotApplicable

Call GDAPI_GetAvailableAttributeValue to determine the color names for a each device media. The following structure is used to retrieve color name information:

```
typedef struct
{
    wchar_t DeviceMediaName[ GDAPI_SHORT_STRING ];
    wchar_t DeviceColor [GDAPI_SHORT_STRING ];

} GDAPI_ColorOrder;
```

- DeviceMediaName
  This is the name of the media returned for the DEVICE_MEDIA attribute.
- DeviceColor
  One device color is returned for each call to GDAPI_GetAvailableAttributeValue.

7.6.3 INVERT_RASTER

The INVERT_RASTER attribute is used to control whether or not the DSI inverts the raster data delivered by GDAPI_SendImageData.

Note that a DSI only has to support the invert operation for half-tone data – con-tone data should be inverted by the raster source if the inversion of con-tone data may also require color adjustments.

DEVICE_ATTRIBUTE values for INVERT_RASTER:

Field	Value
Name	INVERT_RASTER
Type	AttributeType_Bool
Data Scope	DataScope_Separation_n
Raster Source Access	Access_ReadWrite
Selection Model	SelectionModel_NotApplicable

Call GDAPI_SetAvailableAttributeValue to set the invert raster flag for all Print Sessions, a specific Print Session, or for a specific separation.

Data:

```
bool invertRaster;
```

7.7 Standard Attribute Type Overview

This section describes the data types which correspond to the standard attribute types defined in GDAPI_AttributeType, except for AttributeType_Custom which is different for each custom attribute and defined and described for each custom attribute type.

This section also provides key examples to show how attributes of standard types are handled.

Standard attribute types are defined to allow a raster source to deal with attributes of these types in a generic way.

7.7.1 Reading Standard Attributes

Standard attributes must be read one value at a time. This is because:

- the raster source allocates the memory for storing attribute values, and
- an attribute can represent a list of a variable length.

7.7.2 Writing Standard Attributes

Standard attributes must be written in a single call to GDAPI_SetCurrentAttributeValue. This is because an attribute can represent a list of items from which the raster source can select multiple items. Examples are given below.

7.7.3 Example: List Attribute

This example presents a imaginary PUNCH attribute as a list from which the raster source can select one or more values.

Note that a PUNCH attribute will be device dependent – a co-ordinate in some cases or an enumerated list of values, as presented in this example.

DEVICE_ATTRIBUTE values for PUNCH:

Field	**Value**
Name	PUNCH
Type	AttributeType_String_ID
Data Scope	DataScope_PrintSession
Raster Source Access	Access_ReadWrite
Selection Model	SelectionModel_Multiple

Reading:
The values **available** for PUNCH are read by GDAPI_GetAvailableAttributeValue;

each call returns one value in a “string id” field. The **current** value of PUNCH is read by GDAPI_GetCurrentAttributeValue.

Values are returned as GDAPI_StringId values:

```
GDAPI_StringId stringId;
```

The available values returned are:
IDS_PUNCH_NONE, IDS_PUNCH_A, IDS_PUNCH_B, IDS_PUNCH_C, and IDS_PUNCH_D. Note that a locale specific string can be retrieved for each string ID by calling GDAPI_GetLocalizedString.

Writing:
The raster source indicates PUNCH selections to the DSI by using the following standard “string ID list” structure:

```
typedef
{
    long            numElements;
    GDAPI_StringId* element;

} GDAPI_StringIdList;
```

A **multiple selection** of IDS_PUNCH_A and IDS_PUNCH_C is indicated by:

- Setting numElements to 2.
- Assigning “element” with an array of two GDAPI_StringId fields, one for IDS_PUNCH_A and a second for IDS_PUNCH_C.
- Calling GDAPI_SetCurrentAttributeValue with an attribute name of PUNCH and a pointer to the GDAPI_StringIdList structure.

A **single selection** of IDS_PUNCH_A is indicated by:

- Setting numElements to 1.
- Assigning “element” with an array of one GDAPI_StringId fields, one for IDS_PUNCH_A.
- Calling GDAPI_SetCurrentAttributeValue with an attribute name of PUNCH and a pointer to the GDAPI_StringIdList structure.

7.7.4 Example: Non-List Attribute

This example presents the BIT_DEPTH attribute as a numerical range from which the raster source can set a single value.

DEVICE_ATTRIBUTE values for BIT_DEPTH:

Field	Value
Name	BIT_DEPTH
Type	AttributeType_RangeLong

Data Scope	DataScope_PrintSession
Raster Source Access	Access_ReadWrite
Selection Model	SelectionModel_NotApplicable

Reading:
The possible range for BIT_DEPTH is read by GDAPI_GetAvailableAttributeValue; the call returns the range of valid values in the following standard "range long structure":

```
typedef struct
{
    long minValue;
    long maxValue;

} GDAPI_RangeLong;
```

The current bit depth setting is read by GDAPI_GetCurrentAttributeValue; the call returns the current value:

```
long CurrentValue;
```

Writing:
The raster source indicates the BIT_DEPTH setting to the DSI by using a data type of long:

```
long CurrentValue;
```

A selection of 8 is indicated by:

- Setting CurrentValue to 8.
- Calling GDAPI_SetCurrentAttributeValue with an attribute name of BIT_DEPTH and a pointer to the CurrentValue variable.

7.8 Standard Attribute Types

The data types for standard attributes are defined to handle attributes which can be set to a single value and attributes which represent a list where either single values can be selected or multiple values can be selected from the list of possible values.

The following sections describe the data types which correspond to each of the standard attribute types defined in GDAPI_AttributeType:

7.8.1 AttributeType_String

See the introductory sections for examples of handling non-list attributes.

This is the attribute type to use for a locale sensitive string:

- Use stringId to get a localized string from the GDAPI_GetLocalizedString function.

- The stringId can be used by the raster source to store the current value in a “locale **insensitive**” way; the locale specific string can be displayed as required.

Reading:
```
GDAPI_StringId stringId;
```

Writing:
```
GDAPI_StringId stringId;
```

7.8.2 AttributeType_StringList

See the introductory sections for examples of handling list attributes.

This is the attribute type to use for a list of locale sensitive strings:

- Use stringId to get a localized string from the GDAPI_GetLocalizedString function.
- The stringId can be used by the raster source to store the current value in a “locale **insensitive**” way; the locale specific string can be displayed as required.

Reading:
```
GDAPI_StringId stringId;
```

Writing:
```
typedef
{
    long            numElements;
    GDAPI_StringId* element;

} GDAPI_StringIdList;
```

7.8.3 AttributeType_Long

See the introductory sections for examples of handling non-list attributes.

Reading:
```
long CurrentValue;
```

Writing:
```
long CurrentValue;
```

7.8.4 AttributeType_LongList

See the introductory sections for examples of handling list attributes.

Reading:
```
long CurrentValue;
```

Writing:

```
typedef
{
    long  numElements;
    long* element;

} GDAPI_LongList;
```

7.8.5 AttributeType_Double

See the introductory sections for examples of handling non-list attributes.

Reading:

```
double CurrentValue;
```

Writing:

```
double CurrentValue;
```

7.8.6 AttributeType_DoubleList

See the introductory sections for examples of handling list attributes.

Reading:

```
double CurrentValue;
```

Writing:

```
typedef
{
    long    numElements;
    double* element;

} GDAPI_DoubleList;
```

7.8.7 AttributeType_Bool

See the introductory sections for examples of handling non-list attributes.

Reading:

```
bool
```

Writing:

```
bool
```

7.8.8 AttributeType_RangeLong

See the introductory sections for examples of handling non-list attributes.

Reading:

```
GDAPI_GetAvailableAttributeValue returns:

typedef struct
{
    long minValue;
    long maxValue;
```

```
} GDAPI_RangeLong;

GDAPI_GetCurrentAttributeValue returns:

long CurrentValue;
```

Writing:

```
long CurrentValue;
```

7.8.9 AttributeType_RangeDouble

See the introductory sections for examples of handling non-list attributes.

Reading:

```
GDAPI_GetAvailableAttributeValue returns:

typedef struct
{
    double minValue;
    double maxValue;

} GDAPI_RangeDouble;

GDAPI_GetCurrentAttributeValue returns:

double CurrentValue;
```

Writing:

```
double CurrentValue;
```

8. GLOSSARY

Attribute
An attribute of a device such as the media supported, punch options supported, or queue length.

AwayEdge
The edge of the plate where the slow can ends.

Core Attributes
Core Attributes are attributes which are supported across all imaging devices; see the “Constants” section for details.

Device Color Name
The color name used change settings on a device in preparation for imaging that color.

DSI
Device Specific Implementation. A DSI is a GDAPI implementation which knows how to control one or more devices.

FastScan
The resolution around the drum.

HomeEdge
The edge of the plate where the slow scan begins.

Image
Synonymous with separation; the data for one separation.

Image Area
The area on a media which can be imaged; this is typically the area inside the margins described by the MEDIA_SIZE attribute, but can be larger if the DSI or the device allows imaging outside of the margins.

LeadingEdge
The edge of the plate where the fast scan begins.

Negative Media
A pixel value of 0 in the *final* image data (I.E *after* any data inversion) is a colored pixel on the media.

Non-Core Attributes
A Non-Core Attribute is any attribute which is not a Core Attribute.

Positive Media
A pixel value of 0 in the *final* image data (I.E *after* any data inversion) is a transparent pixel on the media.

Separation
Synonymous with image; the data for one separation.

Session
A group of separations to be imaged on a device.

SlowScan
The resolution across the drum.

TrailingEdge
The edge of the plate where the fast scan ends.