1.1 Service Operations

A User makes a Service request by interacting through a local Client (via the MFD console) or through a remote Client via its software application user interface. Each Service provides the same set of service interfaces for the co-located local Client or a Remote Client. The Client can operate via a local interface, a local area network, or the Internet.

The individual Service specifications identify all operations applicable to that Service, some of which may be unique to that Service. The MFD Operations described in this section are common to more than one Service, with the “<service>” component in the operation name identifying the specific Service to which the operation request is addressed. To the extent these operations are implemented by a Service, they MUST conform to the MFD Operation definitions in this section.

All operations consist of a Request issued by a client and a Response returned by the Service. All Requests are sent to the target Service except Startup<service>Service, which is sent to the MFD System. A Service MUST respond to every request addressed to it. Some responses may just indicate that the request was or was not honored, perhaps with explanation messages; others will contain requested or related information, perhaps with additional explanatory “reasons” information.

The MFP operations applicable to two or more Services are listed in Table along with references to their antecedent Printer operations. The operations are, for the most part, derived from IPP Print Service operations defined in RFC2911 [ref], REF3390[ref] and RFC3998 [ref] and further discussed in the PWG IPP 2.0 document [ref] and the IPP Job and Printer Extensions-Set 2 document [ref]. The print specific context has been extended to MFD Services. Several print operations have been omitted as inapplicable to MFD Services as a whole, while a few operations have been added.

Table 1 MFD Common Operations and Antecedents

MDF Operation	Antecedent IPP Operation	Reference		Access
Add<service>HardcopyDocument	[New]			User
Cancel<service>Document	Cancel-Document	[PWG5100.5]	4.5	User
Cancel<service>Job	Cancel-Job	[RFC2911]	3.3.3	User
Cancel<service>Jobs	Cancel-Jobs	[JPS2]	5.1	Admin
CancelCurrent<service>Job	Cancel-Current-Job	[RFC3998]	4.2	User
CancelMy<service>Jobs	Cancel-My-Jobs	[JPS2]	5.2	User
Close<service>Job	Close-Job	[JPS2]	5.3	User
Create<service>Job	Create-Job Print-Job	[RFC2911]	3.1.4 3.2.1	User
Disable<service>Service	Disable-Printer	[RFC3998]	3.1.1	Admin
Enable<service>Service	Enable-Printer	[RFC3998]	3.1.2	Admin
Get<service>DocumentElements	Get-Document-Attributes	[PWG5100.5]	4.3	User
Get<service>Documents	Get-Documents	[PWG5100.5]	3.3	User
Get<service>JobElements	Get-Job-Attributes	[RFC2911]	3.3.4	User
Get<service>JobHistory	Get-Jobs (which-Jobs element = ‘completed’)	[RFC2911]	3.2.6	User[W1]
Get<service>ServiceElements	Get-Printer-Attributes, Get-Printer-Supported-Values	[RFC2911], [RFC3380]	3.2.5	User
GetActive<service>Jobs	Get-Jobs (which-jobs element = ‘not-completed’)	[RFC2911]	3.2.6	User[W2]
Hold<service>Job	Hold-Job	[RFC2911]	3.3.5	User

MDF Operation	Antecedent IPP Operation	Reference		Access
HoldNew<service>Jobs	Hold-New-Jobs	[RFC3998]	3.3.1	Admin
Pause<service>Service	Pause-Printer	[RFC2911]	3.2.7	Admin
Pause<service>ServiceAfterCurrentJob	Pause-Printer-After-Current-Job	[RFC3998]	3.2.1	Admin
Promote<service>Job	Promote-Job Schedule-Job-After	[RFC3998] [RFC3998]	4.4.1	Admin
Release<service>Job	Release-Job	[RFC2911]	3.3.6	User
ReleaseNew<service>Jobs	Release-Held-New-Jobs	[RFC3998]	3.3.2	Admin
Restart<service>Service	Restart-Printer	[RFC3998]	3.5.1	Admin
Resubmit<service>Job	Resubmit-Job	[JPS2]	5.4	User
Resume<service>Job	Resume-Job	[RFC3998]	4.3.2	User
Resume<service>Service	Resume-Printer	[RFC2911]	3.2.8	Admin
Send<service>Document	Send-Document	[RFC2911]	3.3.1	User
Send<service>URI	Send-URI	[RFC2911]	3.3.2	User
Set<service>DocumentElements	Set-Document-Attributes	[PWG5100.5]	4.4	User
Set<service>JobElements	Set-Job-Attributes	[RFC3380]	4.2	User
Set<service>ServiceElements	Set-Printer-Attributes	[RFC3380]	4.1	Admin
Shutdown<service>Service	Shutdown-Printer	[RFC3998]	3.5.2	Admin
Startup<service>Service	Startup-Printer	[RFC3998]	3.5.3	Admin
SuspendCurrent<service>Job	Suspend-Current-Job	[RFC3998]	4.3.1	User

The MFD operation definitions in this section are generic. Depending on the encoding used by the binding, the actual identification of the operation may be different. For example, IPP uses a numeric code. Further, depending on the addressing inherent in the transport, the operation requests might include an implicit rather than explicit identification of the Service. For example, IPP operations coming on the TCP port 631 are inherently Print Service operations.

The MFD Operation definitions are divided between basic or User (Job-oriented) operations and administrative operations. The basic MFD operations are listed in Table 49. These operations are concerned primarily with creating, monitoring, modifying and canceling Jobs and Job-related elements. Basic operations are available to Users including Administrators and Operators, although any operation affecting a Job or Document is restricted to the Job Owner or to an Administrator or Operator. Identification and authentication of the User as Job Owner depends upon the Service and binding, as well as the specific implementation. For example, the Copy Service may consider whoever is present at the machine to be the Job Owner.

Administrative Operations, accessible only to Administrators (and Operators), are concerned primarily with managing the Service and are listed in Table 52. Note that for some Services where the User is present at the device (such as Copy), certain operations may consider any User that is present at the implementing device as having Administrator access.

Table 2 – Basic MFD Interface Requests and Responses

Operation	Request Parameters (Notes 1, 2, 3)	Response Parameters (Notes 1 and 4)	Note
Add<service>HardcopyDocument	InputSource, JobId, DocumentTicket, LastDocument	DocumentNumber, UnsupportedAttributes	
Cancel<service>Document	Target Document		
Cancel<service>Job	JobId		
CancelCurrent<service>Job	JobId (optional)		
CancelMy<service>Jobs	JobIds (optional) Message (optional)		

Operation	Request Parameters (Notes 1, 2, 3)	Response Parameters (Notes 1 and 4)	Note
Create<service>Job	JobTicket (optional)	Job ID	
GetActive<service>Jobs	Limit	JobSummaries including JobID, JobName, JobOriginatingUserName, JobState and perhaps JobStateReasons	
Get<service>DocumentElements	JobId, Document Number, RequestedElements (JobReceipt, JobStatus, or JobTicket)	ElementData, Unsupported Elements, Job Summary	
Get<service>Documents	JobId, RequestedElements (JobReceipt, JobStatus, or JobTicket)	DocumentNumber(s)	
Get<service>JobElements	JobId, RequestedElements (JobReceipt, JobStatus, or JobTicket.)	ElementData	
Get<service>JobHistory	Limit	JobSummaries including JobID, JobName, JobOriginatingUserName, JobState and perhaps JobStateReasons	
Get<service>ServiceElements	RequestedElements (ServiceCapabilities, ServiceConfiguration, ServiceDescription, ServiceStatus or DefaultJobTicket.)	ElementData	
Hold<service>Job	JobId		
Release<service>Job	JobId		
Resubmit<service>Job	JobId, JobTicket (optional)	JobId	
Resume<service>Job	JobId		
Send<service>Document	JobId, DocumentData	DocumentNumber	
Send<service>Uri	JobId, Document URI	DocumentNumber	
Set<service>DocumentElements	JobId, DocumentNumber, RequestedElements & Values		
Set<service>JobElements	JobId, RequestedElements & Values		
SuspendCurrent<service>Job	JobId		

Notes:

Note 1: All operation requests for a response that includes a message must include an ElementsNatuiralLanguageRequested element. All operation requests and responses that include messages must include an ElementsNaturalLanguage element.

Note 2: Table indicates minimum arguments. Client may include additional information. MFD may use this information or not, but must not reject the request if additional information is included.

Note 3: Any request that affects a job or requests information about a specific job must include the RequestingUserName, which is used by the Service to determine whether the requestor is an Administrator, Operator or the Job Owner and is therefore authorized to make the request. Some implementations may require further authentication of the requestor's identity. If the requestor is not determined to have access, the Service MUST reject the request.

Note 4: All responses must include correlation to request and whether request was successful or failed. Table indicates minimum response in addition to success if request was successful. Service may include additional information (such as reason for failure) that the client may use or not, but the client must accept the response whether or not additional information is included.

1.1.1 Basic Service Operations

The common Basic operations are listed in Table 49; they are concerned with creating and controlling Jobs and Documents within Jobs. The Operations include those by which a client gets Service elements to allow selection of Services and formulation of Job Tickets. Some of these operations do affect the state of a Job. ese operations directly affects the state or configuration of the Service except to the extent that creating or canceling a Job may initiate a sequence that affects the Service.

Any Job-oriented basic operation MUST be rejected by a Service if the operation requestor is not the Job owner or an Administrator or Operator; that is,

1.1.1.1 Add<service>HardcopyDocument

The Add<service>HardcopyDocument operation allows a client to prepare a Service to accept a Hardcopy Document via a scanner subunit and to add it to an identified Job. It is analogous to the Send<service>Document and Send<service>Uri operations except that it is applicable to Services for which input Documents are obtained by a scan of a region of a media sheet side, such as FaxOut and EmailOut.

The Service MUST reject this request and send an appropriate message if:

1. The requestor is not the owner of the identified Job, or is not an Administrator or Operator;
2. The Service has already closed inputs to the identified Job, or the Job is not found.

Otherwise, provided the request is properly constructed, complete and references valid objects, the Service MUST accept the request, MUST close the Job if the LastDocument element is asserted, MUST be prepared to add DocumentData from the identified input to the identified Job, and MUST respond to the request.

1.1.1.2 Cancel<service>Document

The Cancel<service>Document operation allows a client to cancel a specified Document in a specified Job of the specificed Service any time from when the time the Document is created up to, but not including, the time that the Document is Completed, Cancelled , or Aborted. Because a Document might already be in Processing by the time a Cancel<service>Document request is received, some portion of the Document processing might be completed before the it is actually terminated.

The Cancel<service>Document operation does not remove the Document from the Job or the Service, but does set the specified Document's DocumentState DocumentStatus element to Cancelled and the Document's DocumentStateReasons element to an appropriate value. If the Job containing the Document is again submitted using Resubmit<service>Job, the canceled Document is also submitted for processing. Thus Cancel<service>Document has the same semantics as Cancel<service>Job which cancels only the processing of the Job but does not delete the Job object itself.

The Cancel<service>Document operation does not affect the states of any of the other Documents in the Job. If the Job is in the Processing state and there are more Documents to be processed, the Service does continue to process the un-canceled Documents. If there are no further Documents to process, the Job is advanced to the Completed state.

The Service MUST reject the operation and return an appropriate response message if the operation requestor is not either the Job owner or a Service or System operator or administrator. Otherwise.the Service MUST accept or reject the Cancel<service>Document request based on the document's current state and, if the request is accepted, the Service MUST transition the Document to the indicated new state as follows:

Table 3 - DocumentState Change by Cancel<service>Document

Initial DocumentState	New DocumentState	Service response
Pending	Cancelled	success
Processing	Cancelled	success
Processing	Processing (if there is a significant delay in transitioning to Cancelled. Also, the DocumentState value must be set to indicate that	success

	the Document is transitioning to Cancelled).	
Processing, but with DocumentState value indicating Document is transitioning to Cancelled .	(Operation has no effect on DocumentState , which may be Processing or Cancelled depending on progress of previously initiated transition)	client error
Completed	Completed	client error
Cancelled	Cancelled	client error
Aborted	Aborted	client error

Once a “success” response has been sent, the implementation guarantees that the Document will eventually end up in the Cancelled state. Between the time that the Cancel<service>Document request is accepted and when the Document enters the Cancelled Document-state, the DocumentStateReasons element MUST contain a value which indicates to any later query that, although the Document might still be Processing, it will eventually end up in the Cancelled state.

1.1.1.3 Cancel<service>Job

The Cancel<service>Job operation changes the state of the identified Job to Cancelled, provided that the Job is not already in or in a mode leading directly to a termination state. (i.e., Completed, Cancelled , or Aborted.) See Table 51. Because a Job might already be active by the time a Cancel<service>Job is received, a portion of the Job may be done before the Job is actually terminated.

The Service MUST accept or reject the request based on the Job's current state. If the request is accepted, the Job state is transitioned to Cancelled and the Service will issue a success response. See transition diagram under Job State. If the implementation requires some measurable time to cancel a Job in the Processing or ProcessingStopped states, the Service MUST set the Job's JobStateReasons to a value indicating that the Job is transitioning to a Cancelled state. If the Job already has a JobStateReasons indicating that it is transitioning to a Cancelled state, then the Service MUST reject a Cancel<service>Job operation.

Table 4 –Legal Transitions Effected by Cancel<service>Job Operation

Current JobState	**Condition**	**New JobState**	**Request Response**	**Note**
Pending	-	Cancelled	Success	
PendingHeld	-	Cancelled	Success	
Processing		Cancelled	Success	
	Implementation takes time to effect cancel.	Processing	Success	JobStateReasons will be set to ProcessingToStopPoint value
	JobStateReasons is set to ProcessingToStopPoint	Processing	Failure	Job already progressing to canceled state
Processing Stopped		Cancelled	Success	
	Implementation takes time to effect cancel.	Processing Stopped	Success	JobStateReasons will be set to ProcessingToStopPoint value
	JobStateReasons is set to ProcessingToStopPoint	Processing Stopped	Failure	Job already progressing to canceled state
Completed		Completed	Failure	Job already terminiated
Cancelled		Cancelled	Failure	Job already terminiated
Aborted		Aborted	Failure	Job already terminiated

1.1.1.4 CancelCurrent<service>Job

The CancelCurrent<service>Job operation allows a client to cause the Service to terminate processing on the currently processing Job and to move that Job to the Cancelled state. As with any other Basic operation directly affecting a Job, this operation is accepted by the Service only if the originator is the Owner of the affected Job(s) or is an Administrator or Operator; that is, the operator requestor has access rights.

There is the potential that the current Job may have changed between the time a client requests this operation and the time the Service implements it. Therefore, if the intent is to suspend a particular Job. the Client MAY include an optional JobID parameter in the request.

1. If the JobID is included in the request and that Job is currently in the Processing or ProcessingStopped state and the operation requestor has access rights to that Job, the Service MUST accept the request and cancel the Job.
2. If no JobId is included in the request and the operation requestor has access rights to the Job currently in the Processing or ProcessingStopped state, the Service MUST accept the request and cancel that Job.
3. If more than one Job is in the Processing or ProcessingStopped state, all currently processing Jobs to which the request originator has access MUST be cancelled unless the operation included the optional JobID, in which case only the identified Job is canceled.
4. If the JobID is included in the request and that Job is not currently in the Processing or ProcessingStopped state; or if the requestor does not have access rights to the identified Job, the Service MUST reject the request and return the appropriate error code.
5. If there is no Job currently in the Processing or ProcessingStopped state or if the requestor doses not have access rights to any Job that is in the Processing or ProcessingStopped state, the Service MUST reject the request and return the appropriate error code.

1.1.1.5 CancelMy<service>Jobs

The CancelMy<service>Jobs operation permits a user to cancel all of their own identified non-Terminated Jobs or, if no specific Jobs are identified in the request, to cancel all of their own non-Terminated Jobs in the Service. This operation works like the Cancel-Job operation except that it can work on multiple Jobs at once. If the Service cannot cancel all of the requested Jobs successfully, it MUST NOT cancel any Jobs and MUST return an error code along with a list of offending JobIds.

The client specifies the set of candidate Jobs to be canceled by supplying and/or omitting the JobIds. The Service MUST check the access rights of the requesting user against *all* of the candidate Jobs. If *any* of the candidate Jobs are not owned by the requesting user, the Service MUST NOT cancel any Jobs and MUST return the appropriate error status code along with the list of un-cancelable JobId values. If this check succeeds, then (and only then) the Service MUST accept or reject the request based on the current state of each of the candidate Jobs and transition each Job to the indicated new state as shown in Table 51. If any of the candidate Jobs cannot be canceled, the Service MUST NOT cancel any Jobs and MUST return the indicated error staus code along with the list of offending JobId values.

1.1.1.6 Close<service>Job Operation

The Close-Job operation allows a client to close Job inputs to those Services accepting Documents, even when the last Document input operation for the Job (Send<service>Document, Send<service>URI or Add<service>Document) did not include the LastDocument element with a 'true' value. This Close<service>Job operation supersedes and, if supported by the Service, is preferable to the practice of using a Send<service>Document with no document data but with a LastDocument element containing a 'true' value to close inputs.

The Service MUST reject this operation request if the target Job is not found or if the requestor is not the Job Owner or an Administrator. Otherwise, the Service MUST accept this operation request even if the target Job is already closed and regardless of JobState. Closing the Job MUST cause the Service to reject any subsequent Document input operation for the target Job, but MUST NOT affect the execution of any previously accepted Document input operation.

1.1.1.7 Create<service>Job

The Create<service>Job operation allows a Client to request creation of a Job in the Service. Upon creation, the Job is in Pending state and available for scheduling unless a Job Processing instruction prevents this. (e.g. JobHoldUntil puts it in PendingHeld state) The Create<service>Job operation MUST fail if the Service's IsAcceptingJobs element value is 'false'.

Job Processing is done on one or more Documents. Unlike the antecedent IPP Print-Job operation, the MFD Create<service>Job may involve more than one Document. Depending upon the type of Service, the input may be a HardcopyDocument or a DigitalDocument. In either case, the source(s) of the input document(s) as well as the destination(s) of the output document(s) are identified in the JobTicket submitted in the Create<service>Job Request,

Once a Job is created, Documents may be input as part of that Job by Send<service>Document, Send<service>URI or, for Services that accept hardcopy input, Add<Service>Document operations. In Service implementations that do not accept multiple documents (i.e., MultipleDocumentJobsSupported = False), document input is closed after one Document is accepted. In Service implementations that do accept multiple documents (i.e., MultipleDocumentJobsSupported = True), there may be multiple Send<service>Document, Send<service>URI or Add<Service>Document operations. There are two methods of indicating when all Documents have been input:

issuing a Close<service>Document request
issuing a Send<service>Document, Send<service>URI or Add<Service>Document request with the LastDocument element = True

To avoid a possible hang condition, Service implementations supporting multiple Document Jobs must also support the MultipleOperationTimeOut element that indicates the minimum number of seconds the Service will wait for the next Send or Add operation before taking some recovery action. If, for some reason, there is a longer period between Create<service>Job and valid Send or Add operations, or between sequential Send or Add operations, the Client MUST send Send or Add requests, even if they are empty, to reset the timeout. If there is a multiple operation timeout, the Service will take remedial action according to the value that Service has indicated in its MultipleOperationTimoutAction element.

1.1.1.8 Get<Service>DocumentElements

The Get<Service>DocumentElements operation allows a Client to obtain detailed information about the specified Document within the specified Job. This operation is parallel to the Get<service>Job-Elements operation, but with the target and response elements relating to a Document rather than a Job.

The Client requests specific groups of elements (complex elements) contained within the Document. The Document Data is not part of the Document and cannot be retrieved using this operation. However the location of the Document Data is available. The allowed values for RequestedElements are DocumentReceipt, DocumentStatus or DocumentTicket. Vendors may extend the allowed values.

The Service MUST return the DocumentDescription element values that a client supplied in the Document Creation operation (Create<service>Job, Send<service>Document or Send<service>URI) or provided in Set<service>DocumentElements operation a plus any additional Document Description elements that the Service has generated, such as DocumentState. The Service MUST NOT return any

Job level elements that the Document object inherits and MUST NOT factor out common Document object elements and return them as Job object elements.

It is NOT REQUIRED that a specific Document include all elements belonging to a group (since some elements are optional). However, it is REQUIRED that the Service support all these group names for the Document object.

1.1.1.9 Get<service>Documents

The Get<service>Documents operation allows a client to retrieve the list of Documents belonging to the target Job. The client MAY also supply a list of the requested Document element names and/or element group names. A group of Document element names with their values will be returned for each Document in the Job.

This operation is similar to the Get<service>DocumentElements operation except that it returns elements from all Documents contained in the Job rather than just the specified Document. As with the Get<service>DocumentElements operation, the Service MUST return only those elements that are in the DocumentTicket

1.1.1.10 Get<service>JobElements

The Get<Service>JobElements operation allows a Client to obtain detailed information on the specified Job. Unlike the antecedent IPP Get-Job-Attributes operation, the Get<Service>JobElements request may not specify individual elements. Rather, the Client requests specific groups of elements contained within the Job. The allowed values for RequestedElements are JobReceipt, JobStatus, or JobTicket. Vendors may extend the allowed values.

The Service MUST reject this request if the requestor is not authorized access to the identified Job,

1.1.1.11 Get<service>JobHistory

The Get<service>JobHistory operation provides summary information on all Jobs that have reached a terminating state (i.e. Completed, Cancelled , Aborted). As such, it is similar to the antecedent Get-Jobs operation with the which-jobs element set to ‘completed’. Unlike Get-Jobs, Get<service>JobHistory may not include a RequestedElements argument; rather, it always returns a JobSummary for each terminated Job including JobID, JobName, JobOriginatingUserName, JobState and perhaps JobStateReasons and other service specific information.

1.1.1.12 Get<service>ServiceElements

The Get<service>ServiceElements operation allows a Client to obtain detailed information on the elements and their values supported by the Service. Unlike the antecedent IPP Get-Printer-Attributes operation, the Get<Service>ServiceElements request may not specify individual elements. Rather, the Client requests information on one or more specific group of elements. The allowed values for Requested Elements are ServiceCapabilities, ServiceConfiguration, ServiceDescription, ServiceStatus or DefaultJobTicket. Vendors may extend the allowed values.

Some Services may accept an additional argument in a Get<service>ServiceElements request to further filter the response, much as the antecedent IPP Get-Printer-Attributes operation accepted the Document-Format element. The individual Service documents identify such arguments if any, their effect and whether support is mandatory.

In addition to the status message, the Service response includes the set of requested element names and their values for all supported elements. The response NEED NOT contain the requested element names for any elements not supported by the Service.

1.1.1.13 GetActive<service>Jobs

The GetActive<service>Jobs operation provides summary information on all Jobs in the Pending or Processing state. As such, it is equivalent to the antecedent Get-Jobs operation with the which-jobs element set to 'not-completed'. Unlike the antecedent Get-Jobs operation, GetActive<service>Jobs may not include a RequestedElements argument; rather, it always returns a JobSummary for each Active Job with the summary including JobID, JobName, JobOriginatingUserName, JobState and perhaps JobStateReasons and other service specific information.

1.1.1.14 Hold<service>Job

The Hold<service>Job operation allows a client to hold a Pending Job in the queue so that it is not eligible for scheduling. The Job Transitions as a result of a HoldJob operation depend upon the current Job state, as indicated in Table 53.

The restraint imposed by a Hold<service>Job is removed by a Release<service>Job operation directed to the same Job. If a Service implementation supports Hold<service>Job, it must also support Release<service>Job and vice-versa.

Table 5 -Transitions Resulting from HoldJob Operation

Current JobState	New JobState	Status	Note
Pending	PendingHeld	Success	See Note 1
Pending	Pending	Success	See Note 2
PendingHeld	PendingHeld	Success	See Note 1
PendingHeld	Pending	Success	See Note 2
Processing	Processing	Failure	
ProcessingStopped	ProcessingStopped	Failure	
Completed	Completed	Failure	
Cancelled	Cancelled	Failure	
Aborted	Aborted	Failure	

Note 1: If the implementation supports multiple reasons for a Job to be in the PendingHeld state, the Server MUST add the JobHoldUntilSpecified value to the Job's JobStateReasons element.

Note 2: If the Service supports the JobHoldUntil and/or the JobHoldUntilTime elements, but the specified time period has already started (or is the NoHold value) and there are no other reasons to hold the Job, the Service MUST make the Job be a candidate for processing immediately by putting the Job in the Pending state.

If the HoldJob operation is supported, then the ReleaseJob operation MUST be supported, and vice-versa. The OPTIONAL JobHoldUntil or JobHoldUntilTime parameter allows a client to specify whether to hold the Job until a specified time, indefinitely or until a specified time period. The Service MUST accept or reject the request based on the Job's current state and transition the Job to the indicated new state as follows. A HoldJob request is rejected when the identified Job is in the Processing or ProcessingStopped states.

1.1.1.15 Release<service>Job

The Release<service>Job operation allows a client to release a previously held Job from the PendingHeld state so that it is eligible for scheduling, provided that there is no other reason to keep the Job in the PendingHeld state. That is, the restraint imposed by a Hold<service>Job operation is removed by a Release<service>Job operation directed to the same Job. If a Service implementation supports Hold<service>Job, it must also support Release<service>Job and vice-versa.

The Job Transitions as a result of a HoldJob operation depend upon the current Job state, as indicated in Table 53.

Table 6 - Job State Transitions Resulting from ReleaseJob Operation

Current Job State	New Job State	Status	Comment
Pending	Pending	success	
Pending-Held	Pending-Held	success	See note below
Pending-Held	Pending	success	
Processing	Processing	success	
Processing- Stopped	Processing- Stopped	success	
Completed	Completed	Failure	
Cancelled	Cancelled	Failure	
Aborted	Aborted	Failure	

Note: If there are other reasons to keep the Job in the PendingHeld state, such as resources nor available, the Job remains in the PendingHeld state. Thus the PendingHeld state is not just for Jobs that have the Job Hold applied to them, but are for any reason that keeps the Job from being a candidate for scheduling and processing.

1.1.1.16 Resubmit<service>Job

The Resubmit<service>Job operation allows a client to resubmit a previously completed Job but with the option of providing new JobTicket information (other than input DocumentData or input DocumentData descriptive information.)

The Resubmit<service>Job operation is applicable only to a retained Job. A retained Job is one which remains in the Service after it has been completed or cancelled. This may be incidentally or because it is a saved Job, which is a Completed or Cancelled Job with a JobSaveDispostion element value that indicates that the Job, including DocumentData if any, should not be deleted or aged-out after the Job is completed.

If a Resubmit<service>Job operation is accepted, the state of the retained Job is not changed; rather, a new Job is created from the identified retained Job and submitted with an implicit CreateJob request.

1. If the Resubmit<service>Job request contains a processing element that was in the retained Job but with a different value, the value supplied in the Resubmit<service>Job operation MUST override the original value (if supported by the Service).
2. If the Resubmit<service>Job request contains a processing element that was not in the retained Job, the element with the value supplied with the Resubmit<service>Job operation MUST be applied (if supported by the Service)
3. For any processing element in the original retained Job the value of which is not changed in the Resubmit<service>Job request, that element and its value MUST be applied to newly created Job except that a JobSaveDispostion element value indicating that the Job should be saved, and certain other Service-specific element values, MUSR NOT be copied but are applied to the new Job only if they are in the Resubmit<service>Job request.

The newly created Job is moved to the Pending or PendingHeld Job state with the same element values as the original saved Job (except for the save element). If any of the documents in the saved Job were passed by reference (Send<service>URI or Send>service>URI), the Service MUST re-fetch the data, since the semantics of Restart<service>Job are to repeat all Job processing. The Service MUST assign new JobUri and JobId values to the newly created Job; the JobDescription elements that accumulate Job progress, such as JobImpressionsCompleted, JobMediaSheetsCompleted, and JobKOctetsProcessed, MUST be an accurate record for the newly created Job.

The Service MUST accept or reject the Resubmit<service>Job Request based on the authority of the requester and the referenced Job's current state. The Requester must either be the Job owner or an operator or administrator of the Service. The target Job must be retained with a Completed or Cancelled state.

1.1.1.17 Resume<service>Job

The Resume<service>Job operation allows a client to resume the identified Job at the point where it was suspended. Provided that no other condition exists that forces the Job to the PendingStopped state, the Service moves the Job from the ProcessingStopped state to the Pending state and removes the JobSuspended value from the Job's StateReasons element. If the identified Job is not in the ProcessingStopped state with the JobSuspended value in the Job's StateReasons element, the Service MUST reject the request and return an appropriate status code, since the Job was not suspended.

If a Service supports Suspend<service>Job or SuspendCurrent<service>Job operations, it MUST support the Resume<service>Job operation, and vice-versa.

1.1.1.18 Send<service>Document

The Send<service>Document operation allows a client to input a DigitalDocument to a Service as part of an already created Job. In response to the Create<service>Job, the Service returns the JobURI and the JobId. For each Document that the client desires to add, the client issues a Send<service>Document request which contains the entire stream of document data for one Document.

If the Service supports this operation but does not support multiple documents per Job, Document input is closed after the first document is accepted and the Service MUST reject subsequent Send<service>Document requests associated with the same Job. Similarly, if the Service does support multiple documents per Job, the Service MUST reject Send<service>Document requests associated with a given Job after inputs to that Job have been closed either a Close<service>Job operation or a previous Send<service>Document with a 'true' value for the LastDocument element. Note that the Client may send and the Service must accept a Send<service>Document request with a 'true' value for the LastDocument element to close input to that Job, even if that request includes no Document data.

See the Create<system>Job description for discussion of issues relating to excessive delay between multiple Send<service>Document requests.

The Service MUST reject this request and send an appropriate message if:

1. The requestor is not the owner of the identified Job, or is not an Administrator or operator
2. The Service has already closed inputs to the identified Job, or the Job is not found.

Otherwise, the Service MUST accept the request, MUST close the Job if the LastDocument element is asserted, MUST add the DocumentData (if any) to the identified Job, and MUST respond to the request

1.1.1.19 Send<service>Uri

The Send<service>URI operation allows a client to input a DigitalDocument to a Service as part of an already created Job. As such, the Send<service>URI operation is identical to the Send<service>Document except that a client supplies a URI reference (DocumentUri element) rather than the document data itself. If a Service supports this operation, clients can use both Send<service>URI and Send<service>Document operations to add new documents to an existing multi-document Job.

As with Send<service>Document, if the Service supports Send<service>URI but does not support multiple documents per Job, the Service MUST reject subsequent Send<service>URI requests associated with the same Job. Similarly, if the Service does support multiple documents per Job, the Service MUST reject Send<service>URI requests associated with a given Job after inputs to that Job have been closed. Job inputs can be closed either by a Close<service>Job operation or a Send<service>Document (NOT a Send<service>URI) request with a 'true' value for the LastDocument element. Note that the Client may send and the Service must accept a Send<service>Document request

with a 'true' value for the LastDocument element to close input to that Job even if that request includes no Document data.

The Service MUST reject this request and send an appropriate message if:

1. The requestor is not the owner of the identified Job, or is not an Administrator or operator
2. The Service has already closed inputs to the identified Job, or the Job is not found.
3. The Service is unable to access the referenced URI

Otherwise, the Service MUST accept the request, MUST close the Job if the LastDocument element is asserted, MUST add the DocumentData (if any) to the identified Job, and MUST respond to the request

See the Create<system>Job description for discussion of issues relating to excessive delay between multiple Send<service>URI requests.

1.1.1.20 Set<service>DocumentElements

The Set<service>DocumentElements operation allows a Client to set the values of identified elements of the specified Document within the specified Job. This operation is parallel to the Set<service>JobElements and Set<service>ServiceElements operations and it follows the same rules for validation, but with the target and response elements relating to a Document rather than a Job or the Service.

The Client must fully identify the elements to be set as well as the set values. The only settable elements are those within the DocumentTicket. The Document Data is not part of the Document and cannot be changed using this operation. If a Document was originally submitted without a given settable element that the Set<service>DocumentElements request attempts to set, the Service adds the specified element to the Document.

If the client identifies a Document element with the out-of-band DeleteElement value, then the Service MUST remove the element and all of its values from the Document. The semantic effect of the client supplying the DeleteElement value in a Set<service>DocumentElements operation MUST be the same as if the element had not originally been supplied with the Document. Any subsequent Get<service>DocumentElements or Get<service>Documents request MUST NOT return any element that has been deleted. However, a client can re-establish such a deleted Document element with any supported value(s) using a subsequent Set<service>DocumentElements operation.

If the client supplies an element in a Set<service>DocumentElements request with the DeleteElement value and that element is not present on the Document object, the Service ignores that supplied element in the request, does not return the element in the Unsupported Elements group, and returns the ‘success’ status code, provided that there are no other problems with the request.

The validation of the Set<service>DocumentElements request is performed by the Service as if the Document had been submitted originally with the new element values (and the deleted elements removed); i.e., all modified Document elements and values must be supported in combination with the Document elements not modified. If such a Document Creation operation would have been accepted, then the Set<service>DocumentElements MUST be accepted. If such a Document Creation operation would have been rejected, then the Set<service>DocumentElements MUST be rejected and the Document MUST be unchanged. In addition, if any of the supplied elements are not supported, are not settable, or the values are not supported, the Service MUST reject the entire operation; the Service MUST NOT partially set some of the supplied elements. In other words, after the operation, all the supplied elements MUST be set or none of them MUST be set, thus making the Set<service>DocumentElements an atomic operation.

The value of JobMandatoryElements supplied in the original Create<service>Job request, if any, MUST have no effect on the behavior of the Set<service>DocumentElements operation. Rather, the Service

must consider that any element or element value in a Set<service>DocumentElements operation is mandatory. The Service MUST reject any request to set a Document element to an unsupported value or to a value that would conflict with another Document element value.

The Service MUST accept or reject the Set<service>DocumentElements operation when the Document's DocumentState element has the values shown in Table 6. Although the Document's current state affects whether the Service accepts or rejects the Set<service>DocumentElements request, the operation MUST NOT change the state of the Document object (since the Document is a passive object and the Document state is a subset of the JobState). For example, if the operation creates a request for unavailable resources, the Job (but not the Document) transitions to a new state.

1.1.1.21 Set<service>JobElements

The Set<Service>JobElements operation allows a Client to set the values of identified elements of the specified Job. The Client must fully identify the elements to be set as well as the set values. In the response, the Service returns success or rejects the entire request with indications of which element or elements could not be set to the specified values.

This operation is parallel to the Set<service>DocumentElements and Set<service>ServiceElements operations and it follows the same rules for validation, but with the target and response elements relating to a Job rather than a Document or the Service

If the client identifies a Job element with the out-of-band DeleteElement value, then the Service MUST remove the element and all of its values from the Job. The semantic effect of the client supplying the DeleteElement value in a Set<service>JobElements operation MUST be the same as if the element had not originally been supplied with the Job. Any subsequent Get<service>JobElements or Get<service>Jobs request MUST NOT return any element that has been deleted. However, a client can re-establish such a deleted Job element with any supported value(s) using a subsequent Set<service>JobElements operation.

If the client supplies an element in a Set<service>JobElements request with the DeleteElement value and that element is not present on the Job object, the Service ignores that supplied element in the request, does not return the element in the Unsupported Elements group, and returns the ‘success’ status code, provided that there are no other problems with the request.

The validation of the Set<service>JobElements request is performed by the Service as if the Job had been submitted originally with the new element values (and the deleted elements removed); i.e., all modified Job elements and values must be supported in combination with the Job elements not modified. If such a Job Creation operation would have been accepted, then the Set<service>JobElements request MUST be accepted. If such a Creation operation would have been rejected, then the Set<service>JobElements MUST be rejected and the Job MUST be unchanged. In addition, if any of the supplied elements are not supported, are not settable, or the values are not supported, the Service MUST reject the entire operation; the Service MUST NOT partially set some of the supplied elements. In other words, after the operation, all the supplied elements MUST be set or none of them MUST be set, thus making the Set<service>JobElements an atomic operation.

The value of JobMandatoryElements supplied in the original Create<service>Job request, if any, MUST have no effect on the behavior of the Set<service>JobElements operation. Rather, the Service must consider that any element or element value in a Set<service>JobElements operation is mandatory. The Service MUST reject any request to set a Job element to an unsupported value or to a value that would conflict with another Job element value.

The Service MUST accept or reject the Set<service>JobElements operation when the Job's JobState element has the values shown in Table 6. Although the Job's current state affects whether the Service accepts or rejects the Set<service>JobElements request, the operation MUST NOT change the state of the Job object (since the Job is a passive object and the Job state is a subset of the JobState). For

example, if the operation creates a request for unavailable resources, the Job (but not the Job) transitions to a new state.

1.1.1.22 SuspendCurrent<service>Job

The SuspendCurrent<service>Job operation allows a Client to suspend a Job by setting a condition in a Job that is currently in the Processing or ProcessingStopped state. This condition, reflected by the JobSuspended value in that Job's JobStateReasons element, causes that Job to be in the ProcessingStopped state. The Service is able to processes other Jobs normally, provided that no other inhibiting conditions exist. Note that a Job may be ProcessingStopped state for other reasons and that, once it has been suspended, the Job will remain in the ProcessingStopped state even after the other conditions have been removed.

There is the potential that the current Job may have changed between the time a client requests this operation and the time the Service implements it. Therefore, if the intent is to suspend a particular Job. the Client can include an optional JobID parameter in the request.

The target Job is:

a) The Job identified by the JobId, if included in the request
b) If the JobId is not included in the request, any Jobs in the Processing or ProcessingStopped state to which the requestor has access rights.

The Service MUST reject the request and send an appropriate message if:

1. There is no target Job in the Processing or ProcessingStopped state to which the requestor has access rights.
2. The target Job or all potential target Jobs have already been suspended.

The Service MUST accept the request, cancel any target Job(s) that have not been previously suspended, and return an appropriate message if:

1. The target JobID is included in the request and that Job is currently in the Processing or ProcessingStopped state (but is not suspended), and the requestor has access rights,
2. If no JobId is included and the requestor has access rights to the Job that is currently in the Processing or ProcessingStopped state (but is not suspended), the Service MUST accept the request and suspend that Job.
3. If more than one Job is in the Processing or ProcessingStopped state (but are not suspended), all such Jobs MUST be suspended unless the operation request included the optional JobID, in which case only the identified target Job MUST be suspended.
4. If the JobID is included in the request and that Job is not currently in the Processing or ProcessingStopped state; or if the JobID is not included and there is no Job currently in the Processing or ProcessingStopped state, the Service MUST reject the request and return the appropriate error code.
5. If the JobID is included in the request and that Job has been suspended; or if no JobId is included and is currently in the Processing or ProcessingStopped state, the Service MUST reject the request and return the appropriate error code.

The Resume<service>Job operation releases a suspended Job. If a Service supports SuspendCurrent<service>Job operation, it MUST support the Resume<service>Job operation, and vice-versa.

1.1.2 Administrative Service Operations

Administrative Service operations directly affect the Service as a whole or affect the jobs of multiple JobOwners. Access is reserved for Administrators or Operators. The MFD Administrative Service Operations are listed in Table 52 and are described below.

Table 7 - Administrative Operations

Operation	Request Parameters (Notes 1, 2, 3)	Response Parameters (Notes 2, 3, 4)	Note
Cancel<service>Jobs	JobIDs (optional)	JobIds of identified but un-cancellable Jobs	
Disable<service>Service			
Enable<service>Service		-	
Hold<service>Job	Job ID; JobHoldUntil or JobHoldUntilTime (optional)	-	
HoldNew<service>Jobs	JobHoldUntil or JobHoldUntilTime (optional)		
Pause<service>Service	-		
Pause<service>ServiceAfterCurrentJob	-		
Promote<service>Job	Job ID (Target) Predecessor JobID(optional)		
Release<service>Job	Job ID		
ReleaseNew<service>Jobs			
Restart<service>Service			
Resume<service>Job	JobID		
Resume<service>Service			
Set<service>ServiceElements	Target elements,		
Shutdown<service>Service			5
Startup<service>Service			

Notes:

Note 1:Table indicates minimum arguments. Client may include additional information. MFD may use this information or not, but must not reject the request if additional information is included.

Note 2: All requests must include the RequestingUserName, which is used by the Service to determine whether the requestor is an Administrator or Operator and is therefore authorized to make the request. Some implementations may require further authentication of the requestor's identity. If the requestor is not determined to have access, the Service MUST reject the request.

Note 3: All operation requests for a response that includes a message must include an ElementsNatuiralLanguageRequested element. All operation requests and responses that include messages must include an ElementsNaturalLanguage element.

Note 4: All responses must include correlation to request and whether request was successful or failed. Table indicates minimum response in addition to success if request was successful. MFD may include additional information (such as reason for failure) that the client may use or not. But the client must accept the response whether or not additional information is included.

Note 5: Forcing the Service state may also force the state of any active Jobs to Aborted.

1.1.2.1 Cancel<service>Jobs

The Cancel<service>Jobs operation allows the Operator or Administrator of the Service to cancel all identified non-Terminated Jobs or, if no specific Jobs are identified in the request, to cancel all non-Terminated Jobs in the Service. It differs from the Cancel<service>Job operation in that it works on a number of Jobs at once. If, following the Legal Job state Transitions in Table, the Service cannot cancel all explicitly or implicitly requested Jobs successfully, it MUST NOT cancel any Jobs but MUST return an error code along with the list of JobIds for those Jobs that could not be cancelled.

The set of candidate Jobs to be canceled is specified by the supplied JobIds. If no JobIds are supplied, it is implicit that all Jobs that are not in a Terminating state are to be canceled. As with all Administrative operations, the Service MUST check the access rights of the requesting user. Provided that the requester has access rights, the Service MUST check the current state of each of the candidate Jobs. If any of the candidate Jobs cannot be canceled, the Service MUST NOT cancel any Jobs and MUST return the indicated error status code along with the list of offending JobId values. If there are no Jobs that cannot be cancelled, the Service MUST transition each identified Job to the indicated new state as shown in Table 51.

Table x –Legal Transitions Effected by Cancel<service>Jobs Operation

Current JobState	Condition	New JobState	Request Response	Note
Pending	-	Cancelled	Success	
PendingHeld	-	Cancelled	Success	
Processing		Cancelled	Success	
	Implementation takes time to effect cancel.	Processing	Success	JobStateReasons will be set to ProcessingToStopPoint value
	JobStateReasons is set to ProcessingToStopPoint	Processing	Failure	Job already progressing to canceled state
Processing Stopped		Cancelled	Success	
	Implementation takes time to effect cancel.	Processing Stopped	Success	JobStateReasons will be set to ProcessingToStopPoint value
	JobStateReasons is set to ProcessingToStopPoint	Processing Stopped	Failure	Job already progressing to canceled state
Completed		Completed	Failure	Job already terminiated
Cancelled		Cancelled	Failure	Job already terminiated
Aborted		Aborted	Failure	Job already terminiated

1.1.2.2 Disable<service>Service

The Disable<service>Service operation prevents the Service from creating any new Jobs by negating the IsAcceptingJobs element. This operation has no effect upon the Service State and the Service is still able to process operations other than Create<service>Job. All previously created or submitted Jobs and all Jobs currently processing continue unaffected.

If the requestor is determined to have proper access, the Service MUST accept this request and MUST negate the IsAcceptingJobs element.

The IsAcceptingJobs element value is reaffirmed by the Enable<service>Service operation. If an implementation supports Disable<service>Service it must also support Enable<service>Service and vice-versa.

1.1.2.3 Enable<service>Service

The Enable<service>Service operation asserts the IsAcceptingJobs element to allow the Service to accept new Create<service>Job requests. The operation has no effect upon the Service State or any other operation requests the Service may receive.

If the requestor is determined to have proper access, the Service MUST accept this request and MUST assert the IsAcceptingJobs element. The Service MUST then be able to accept and implement Create<Service>Job requests, provided that no other inhibiting condition exists.

If a Service implementation supports Disable<service>Service it must also support Enable<service>Service and vice-versa.

1.1.2.4 Hold<service>Job

The Hold<service>Job operation allows a client to hold a Pending Job in the queue so that it is not eligible for scheduling. The Job Transitions as a result of a HoldJob operation depend upon the current Job state, as indicated in Table 53.

If the HoldJob operation is supported, then the ReleaseJob operation MUST be supported, and vice-versa. The OPTIONAL JobHoldUntil or JobHoldUntilTime parameter allows a client to specify whether to hold the Job until a specified time, indefinitely or until a specified time period. The Service MUST accept or reject the request based on the Job's current state and transition the Job to the indicated new state as follows. A HoldJob request is rejected when the identified Job is in the 'Processing' or 'ProcessingStopped' states.

Table 8 -Transitions Resulting from HoldJob Operation

Current JobState	New JobState	Status	Note
Pending	PendingHeld	Success	See Note 1
Pending	Pending	Success	See Note 2
PendingHeld	PendingHeld	Success	See Note 1
PendingHeld	Pending	Success	See Note 2
Processing	Processing	Failure	
ProcessingStopped	ProcessingStopped	Failure	
Completed	Completed	Failure	
Cancelled	Cancelled	Failure	
Aborted	Aborted	Failure	

Note 1: If the implementation supports multiple reasons for a Job to be in the PendingHeld state, the Server MUST add the 'JobHoldUntil' value to the Job's JobStateReasons element.

Note 2: If the Service supports the JobHoldUntil and/or the JobHoldUntilTime elements, but the specified time period has already started (or is the 'NoHold' value) and there are no other reasons to hold the Job, the Service MUST make the Job be a candidate for processing immediately by putting the Job in the 'Pending' state.

1.1.2.5 HoldNew<service>Jobs

The HoldNew<service>Jobs operation allows a client to prevent any new Jobs from being eligible for scheduling by forcing all newly-created Jobs to the PendingHeld state with a JobHoldUntil or JobHoldUntilTime Job Processing element added, depending upon the element supplied with the HoldNew<service>Jobs operation request. The operation has the same effect as a Hold<service>Jobs operation except that any Jobs in the Pending or Processing state when the HoldNew<service>Jobs request is accepted are allowed to go to completion, provided that no other conditions or operations prevent this.

The JobHoldUntil parameter allows a client to specify holding new Jobs indefinitely or until a specified named time period. The JobHoldUntilTime parameter allows a client to hold new Jobs until a specified time. Provided that the requestor is authorized and the operation and requested parameters are supported, a Service MUST accept a HoldNew<service>Jobs request and MUST add the supplied 'JobHoldUntil' or JobHoldUntilTime element to the Jobs. This HoldNew<service>Job condition may be cleared by a ReleaseNew<Service>Jobs operation.

If the HoldNewJobs operation is supported, then the ReleaseNew<Service>Jobs operation MUST be supported, and vice-versa

1.1.2.6 Pause<service>Service

The Pause<service>Service operation allows a client to send the Service to the Stopped state. In this Service state, the Service MUST NOT advance any Job to Job Processing state. Depending on implementation, the Pause operation MAY also stop the Service from continuing to process any current Job, sending the Job to the ProcessingStopped state. That is, depending upon implementation, any Job that is currently in the Processing state is sent to the ProcessingStopped state as soon as the implementation permits; or the Job continues to a termination state as determined by other conditions. The Service MUST still accept CreateJob operations to create new Jobs, provided that there are no other conditions preventing it.

If the Pause operation is supported, then the Resume operation MUST also be supported, and vice-versa.

Service State transitions resulting from a Pause operation are identified in Table 54. Pause implementation should be done as soon as the possible after the request is accepted. If the implementation will take more than negligible time to stop processing (perhaps to finish processing the current Job), the Service may remain in the ‘Processing’ state but MUST add the 'MovingToPaused' value to the Service’s StateReasons element. When the Service transitions to the 'Stopped' state, it removes the 'MovingToPaused' value and adds the 'Paused' value to the Service’s StateReasons element. If the implementation permits the current Job to stop in mid processing, the Service transitions directly to the ‘Stopped’ state with the Service’s StateReasons element set to the 'Paused' value and the current Job transitions to the 'ProcessingStopped' state with the JobStateReasons element set to the 'Stopped' value.

For any Jobs in the 'Pending' or 'PendingHeld' state, the ‘Stopped' value of the Jobs' JobStateReasons element also applies. However, the Service NEED NOT update those Jobs' JobStateReasons element and need only return the 'Stopped' value when those Jobs are queried (so-called lazy evaluation).

Provided that the requestor is authorized, the Service MUST accept the Pause<service>Service request in any Service state and, if so indicated, transition the Service to the indicated new State before responding as follows:

Table 9 - Transitions Resulting from Pause Operation

Current Service State	New Service State	StateReason	Status	Notes
Idle	Stopped	Paused	Success	
Processing	Processing	MovingToPaused	Success	See Note 1
Processing	Stopped	Paused	Success	See Note 2
Stopped	Stopped	Paused	Success	

Note 1: Implementations that do not stop processing of the current Job respond as indicated. When the current Job has entered a termination state and processing is stopped, the Service State goes from ‘Processing’ to ‘Stopped’ and the ‘StateReason value goes from ‘MovingToPaused’ to ‘Paused’.

Note 2: In instances where there is no current Job in the Processing state, and in implementations that are able to pause the current Job, the Service goes immediately to the ‘Stopped’ state with ‘StateReason ‘Paused’ value. In the latter case, the current Job goes to the ‘ProcessingSTopped state with a JobStateReasons element value of ‘Stopped’.

1.1.2.7 Pause<service>ServiceAfterCurrentJob

The Pause<service>ServiceAfterCurrentJob operation allows a client to stop the Service from processing any Jobs once any Job currently in Processing is completed . This operation has no effect on the current Job and the Service MUST complete the processing of the current Job, provided that no other condition or operations preclude it. The Service MUST still accept CreateJob operations to create new Jobs, but MUST prevent any Jobs from entering the 'Processing' state. If the

Pause<service>ServiceAfterCurrentJob operation is supported, then the Resume<service>Service operation MUST also be supported.

Service State transitions resulting from a Pause<service>ServiceAfterCurrentJob operation are identified in Table 55. Note that the response to the Pause<service>ServiceAfterCurrentJob request and the Pause<service>Service request are exactly the same in implementations where the Service implementation is not able to pause a Job currently in the Processing state.

If the implementation will take more than negligible time to finish processing the current Job, the Service will remain in the Processing state and must add the 'MovingToPaused' value to the Service's StateReasons element. When the Service transitions to the 'Stopped' state, it removes the 'MovingToPaused' value and adds the 'Paused' value to the Service's StateReasons element.

For any Jobs in the 'Pending' or 'PendingHeld' state, their state is unchanged but the JobStateReasons element must be set to the 'Stopped' value. However, the Service NEED NOT update those Jobs' JobStateReasons element and only need return the 'Stopped' value when those Jobs are queried (so-called lazy evaluation).

Provided that the requestor is authorized, the Service MUST accept the request in any Service state and MUST transition the Service to the indicated new State as follows before returning the operation response.

Table 10 –System States Changes in Response to Pause<system>SystemAfterCurrentJob Operation

Current Service State	New Service State	StateReason	Status	
Idle	Stopped	Paused	Success	
Processing	Processing	MovingToPaused	Success	See Note 1
Stopped	Stopped	Paused	Success	

Note : Once the currently processing Job completes, the Service state will transition to 'Stopped' and the MovingToPaused StateReason will be remove and replaced with 'Paused'

1.1.2.8 Promote<service>Job

The Promote<service>Job operation schedules the identified Job to be processed next, after the currently processing Job or, if the request includes the predecessor JobID, immediately after the identified predecessor Job. The Promote<service>Job operation is a combination of the IPP Promote-Job and Schedule-Job-After operations. If the predecessor Job is not specified, it acts in the same way as the antecedent IPP Promote-Job operation. If the predecessor Job is specified, it acts the same way as the antecedent IPP Schedule-Job-After operation.

The identified target Job must be in the 'Pending' state. If the identified target Job is not in the 'Pending' state or if the predecessor Job is identified and it is not in the 'Pending', 'Processing' or 'ProcessingStopped' state, the Service MUST reject the request and return an appropriate status code. If the Promote<service>Job request is accepted, the target Job MUST be processed immediately after the current or identified predecessor Job reaches a Termination state (Cancelled, Completed or Aborted)

Note that the action of this operation is consistent even if a previous Promote<service>Job Request has caused some other Job to be scheduled after the current or predecessor Job; that is, within the rescheduling time limitations of the Service, the Job identified in the last Promote<service>Job Request accepted will be processed next.

1.1.2.9 ReleaseNew<service>Jobs

The ReleaseNew<service>Jobs operation allows a client to remove the condition initiated by HoldNew<service>Jobs and to release all Jobs previously forced to a PendingHeld state by the HoldNew<service>Jobs initiated condition so that these Jobs are eligible for scheduling. This is done by remove the 'JobHoldUntilSpecified' value from the Job's JobStateReasons element and changing the Jobs' states to 'Pending'.

Provided that the requestor is authorized, the Service MUST accept this request in any Service state and the Service MUST remove the 'JobHoldUntilSpecified' value from the Job's JobStateReasons element for any Job previously forced to a PendingHeld state by the HoldNew<service>Jobs initiated condition.

If the ReleaseNewScanJobs operation is supported, then the HoldNewScanJobs operation MUST be supported, and vice-versa.

1.1.2.10 Restart<service>Service

The Restart<service>Service operation causes a Service in any state, even a previously shut down instance of a Service, to be initialized and set to the Idle state, provided that no errors occur or conditions exist that would prevent normal operation. The handing of Jobs that were in the Processing, Pending, PendingHeld, and ProcessingHeld states state prior to Restart is implementation dependent, but a Service Restart MUST be performed as gracefully as possible and in a way preserving the content and integrity of any non-terminated Jobs. Job history data, if supported, SHOULD also be preserved.

Provided that the requestor is authorized, the Service MUST accept the request Restart<service>Service regardless of its current state. Providing that no conditions exist that would normally prevent these actions, the Service MUST initialize its State to Idle, clear the StateReasons element and set the IsAcceptingJobs element to true.

1.1.2.11 Resume<service>Service

The Resume<service>Service operation allows a client to cause the Service to resume scheduling Jobs after scheduling has been paused. Provided that the requestor is authorized and the Service supports this operation, a Service MUST accept a Resume<service>Service request regardless of the current Service state; see table . If there are no other reasons why the Service is in the Stopped state, this operation returns the Service from the Stopped state to the Idle or Processing state from which it was paused, and removes the 'Paused' value to the Service's StateReasons element.

If the Resume<service>Service operation is supported, then the Pause<service>Service operation MUST be supported, and vice-versa.

Table 11 - System State Changes in Response to Resume<service>Service

Current Service state	New Service state-	response	Comment
Idle	dle	success	
Processing	Processing	success	
Stopped	Processing	success	If there are Jobs to be processed
Stopped	Idle	success	If there are no Jobs to be processed.
Stopped	Stopped	success	If other conditions causing a Stopped state exist

1.1.2.12 Set<service>ServiceElements

The Set<service>ServiceElements operation allows a Client to set the values of identified elements in the Service, provided that they are settable. Settable Elements may be in ServiceCapabilities, ServiceConfiguration, ServiceDescription and DefaultJobTicket but not in ServiceStatus.

The Service MUST reject the entire request with indications of which element or elements could not be set if a client request attempts to:

1. Set a non-settable element (including an element not in the ServiceCapabilities, ServiceConfiguration, ServiceDescription or DefaultJobTicket groups, a read-only element, and an element not supported or not supported as a writable element in the specific Service implementation)
2. Set a settable element to an invalid value or to a value that conflicts with the values of other Service elements, including elements being set in the same request.
3. Set a greater number of elements in one operation than are supported by the Service implementation (a Service implementation Need Not support set of more than one element at a time)

If there is no reason to reject setting all of the specified elements to the specified values, the Service MUST accept this operation request when it is in the Idle or Stopped state, and SHOULD accept the request when it is in the Processing state.

If the Service accepts the request, only those elements specified in the request are changed unless the definition of one or more of the set elements explicitly specifies an effect upon some other element.

1.1.2.13 Shutdown<service>Service

The Shutdown<service>Service operation forces the Service to the 'Down' state from any state that it is in, in an orderly manner. That is, the Service MUST stop accepting any further client requests, and MUST stop scheduling Jobs for processing as soon as the implementation allows, although it SHOULD complete the processing of any currently processing Jobs. Once down, the Service will no longer respond to any Client requests other than Restart<service>Service request. Unlike the antecedent IPP Shutdown-Printer operation which requires that no Jobs be lost, the disposition of any Jobs in the Service that are in the Pending, PendingHeld, or ProcessingHeld states is implementation dependent. However, as with Restart<service>Service, Service shutdown must be performed as gracefully as possible and in a way in preserving the content and integrity of any non-terminated Jobs. Job history data, if supported, SHOULD also be preserved.

Once shut down, a Service can be roused from its Down state by a Startup<service>Service operation or a Restart<service>Service operation. If a Service implementation supports Shutdown<service>Service it must also support Startup<service>Service and vice-versa.[W3]

Provided that the requestor is authorized, the Service MUST accept this operation and following an orderly progression, transition to the Down state regardless of the current state of the Service.

1.1.2.14 Startup<service>Service

The Startup<service>Service operation is sent to the MFD System and causes a new instance of the specified Service to begin initialization and then to move through the Down state to the Idle state, provided that no errors occur or conditions exist that would prevent normal operation.

If a Service implementation supports Shutdown<service>Service it must also support Startup<service>Service and vice-versa.[W4]