Salutation Architecture Specification (Part-2)

Version 2.0c

June 01, 1999

Limitation of Liability

Even though the members of the Salutation Consortium have reviewed this Specification, the Consortium shall not make any warranty or representation, neither express or implied, with respect to this Specification, its quality or accuracy and it specifically disclaims the warranties of merchantability and fitness for a particular purpose.

No representation of third party rights

The Salutation Consortium makes no representation or warranty whatsoever with regard to the Consortium member or third party ownership, licensing or infringement/non-infringement of intellectual property rights. Each user of this Specification, whether or not a Consortium member, should seek the independent advice of legal counsel with regard to any possible violation of third party rights.

Trademarks

ESC/P (Epson Standard Code for Printers) is a trademark of EPSON Co.

IPDS (Intelligent Printer Data Stream) is a trademark of International Business Machines Corp.

LIPS (LBP Image Processing System) is a trademark of Canon Inc.

Microsoft Windows is a trademark of Microsoft Corporation.

NetWare is a trademark of Novell, Inc.

PAGES (Page Printer Advanced Graphics Escape Set) is a trademark of IBM Japan.

PCL (Printer Control Language) is a trademark of Hewlett-Packard Co.

PJL (Printer Job Language) is a trademark of Hewlett-Packard Co.

PostScript is a trademark of Adobe Systems Inc.

RPDL (Ricoh PDL) is a trademark of Ricoh Corp.

Sun RPC is a trademark of Sun Microsystems, Inc.

SCSA (Signal Computing System Architecture) is a trademark of Dialogic.

TSAPI (Telephony Services API) is a trademark of Novell, Inc.

Versit is a trademark of Apple Computer, Inc., International Business Machines Corp., Lucent Technologies, and Siemens Rolm Communications Inc.

Preface

The Part-1 of the Salutation Architecture Specification document defines the general framework of the architecture and the details of the Salutation Manager Protocol.

The Part-2, this document, consists of the following:

- The **Salutation Personality Protocol** of Functional Units, i.e. the format and protocol of messages, is defined in Chapters 1 through 4.
- The **Attributes** of each Functional Unit are defined. The Attributes definition is included in Chapters 2 through 4.
- Appendix (Chapters 5 and above) contains the definition of values, data types and syntax of all the defined protocol data units except [Fax Data] Functional Unit.

The Part-3 defines the criteria of the Conformance to the Salutation Architecture Specification.

Revision

Version 2.0 (December 02, 1996)

Public release of the final version 2.0 specification part 2.

Version 2.0a, 2.0b

No change and no release

Version 2.0c (June 01, 1999)

Added some attributes required for implementation and corrected the minor errors of the specifications.

Table of Contents

1.Common Framework

One of the characteristics of the Salutation Architecture is the provision of an common framework for service request commands and protocols which is independent of the type of service and data format. The common framework provides the following advantages:

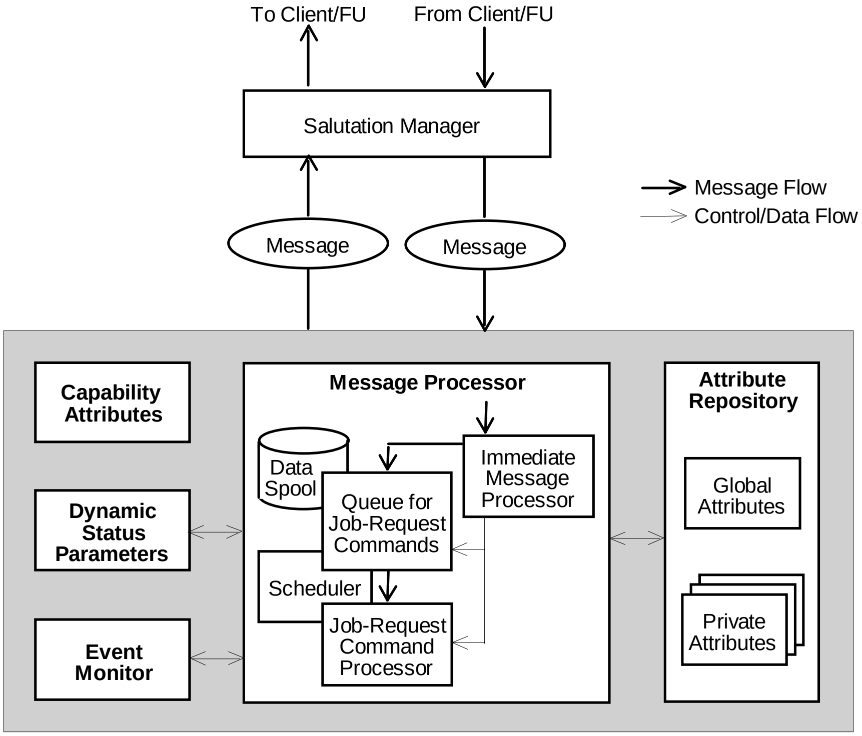
- Easy to learn interface specifications
- Easy to implement coordinated functions
- Easy to expand architecture (easy to add new services)

This chapter describes the characteristics of **Salutation Personality Protocol** that are common across Functional Units.

1.1.Virtual Model of Functional Unit under Salutation Personality Protocol

A Functional Unit exchanges messages with a client, which is either a Salutation client application or another Functional Unit, through a Service Session. When a Service Session is opened by *Open Service*, the protocol to be used in the session is specified by the Personality Protocol ID parameter in the *Open Service* request. If the Salutation Personality Protocol is specified at *Open Service*, messages exchanged between the client application and the Functional Unit or between the two Functional Units in the session follow the definition of this part-2 of the architecture specification document.

The following figure shows the virtual model of a Functional Unit under the Salutation Personality Protocol.



Functional Unit (under Salutation Personality Protocol)

- Capability Attributes

  The Capability Attributes describe the detail of services the Functional Unit can provide. When the Functional Unit registers its capability to the Salutation Manager, it gives the Salutation Manager a Functional Unit Description Record that contains the Capability Attributes. The values of Capability Attributes are static and do not change. A client may query the values of Capability Attributes by issuing a *Query Capability*.

- Dynamic Status Parameters

  Each Dynamic Status Parameter describes an aspect of the current Functional Unit status. The values of Dynamic Status Parameters are dynamic and change. A client may query the current value of a particular Dynamic Status Parameter by sending a QueryDynamicStatus command.

- Event Monitor

  A client may request the Functional Unit to notify the client of any changes of a particular Dynamic Status Parameter. Making such a request is called "subscribing to an event". The Event Monitor monitors the values of subscribed Dynamic Status Parameters, and generates an "event" when the value changes. The "event" is notified to the client by a NotifyEvent message.

- Message Processor

  The Message Processor receives all the messages for the Functional Unit. It processes message by message, and if necessary, builds a message to be sent back to the client.

It is also notified of the initiation and termination of Service Sessions (*Open Service* ~ *Close Service*), and performs session-related housekeeping tasks as required.

Depending on the received messages, the Message Processor operates on the Attribute Repository.

Some messages that take relatively long time to process are not executed immediately. For example, if a job-request-type command (see "Job-Related Messages" section on page 33) is received, the Message Processor performs the following tasks:

1) Creates a job instance
2) Assigns a JobHandle
3) Builds a response to return the JobHandle to the client. It is sent to the client either at this step or at the end of the next step depending on the received command.
4) If the job-request-type command indicates that associated data item(s) must be obtained immediately, the Message Processor initiates Data Transfer Message Sequence (see “Data Transfer Messages” section on page 18) to get the data item(s). (Note that data transfer occurs either when the job-request-type command is received as described here, or when the job-request-type command is actually executed, depending on the command or its parameters.)
5) Enqueues the job-request-type command and, if present, the associated data item(s) in the Job Queue and the Data Spool respectively. If necessary, default attribute values are assigned to the command from the attributes in the Attribute Repository at this stage. See the description of Attribute Repository below for the detail.

The queued commands will be processed by the Queued Job-Request Command Processor in turn. The Scheduler determines which queued command is to be processed next, by inspecting the queued commands. Some Functional Units do not require, or allow optional implementation of the Queue and the Data Spool capability.

- Attribute Repository

An **attribute** in the Attribute Repository is a parameter which controls a way services are done by the Functional Unit. It is sometimes called "**Command Attribute**" to discriminate it from a Capability Attribute. The specification of each Functional Unit defines all the attributes associated with the Functional Unit.

The Attribute Repository contains the following two types of attributes.

□ **Global Attribute**

A single image of the Global Attributes is shared by all the clients of the Functional Unit.

Global Attributes are read only.

The value of each Global Attribute is implementation dependent.

A typical use of Global Attribute is to set default parameter values for the equipment by the administrator.

□ **Private Attribute**

Private Attributes are private to a client, and not shared by the other clients. A separate image of the Private Attributes is associated with each client of the Functional Unit.

Private Attributes are settable (read/write).

Initially, there is no Private Attribute in the Attribute Repository.

A typical use of Private Attribute is to set default parameter values that are applied to subsequent commands.

Private Attributes values are valid only until the current Service Session (*Open Service* ~ *Close Service*) is terminated.

The proper use of Global Attribute and Private Attribute can provide a flexible way of specifying an attribute value, as follows:

1) The attribute value specified in the command is used, if present.

2) Otherwise, the attribute value specified in the Private Attribute is used, if present.

3) Otherwise, the attribute value specified in the Global Attribute is used. (If the attribute is not defined as a Global Attribute, the architecture-defined default value is used. An example is the “printCopyCount” attribute of the [Print] Functional Unit.)

In the cases of 2) or 3), the default attribute value is assigned from the Private Attribute or the Global Attribute respectively to each command when the command is received by the Functional Unit. It is NOT that the value of the Private/Global Attribute is used at the time the command is executed. Therefore, in the following example, the *COMMAND* uses the value "aaa" for XYZ attribute.

Client	**Server**
SetPrivateAttribute({XYZ="aaa"}) =>	
	<= ACK(NULL)
COMMAND(XYZ attribute is omitted) =>	
	The server enqueues the *COMMAND*.
:	
SetPrivateAttribute({XYZ="bbb"}) =>	
	<= ACK(NULL)
:	
	The server dequeues the *COMMAND* and execute it. (XYZ="aaa" is used.)

1.2.Message

The Functional Unit receives a command from a client application or another Functional Unit (called the client hereafter), and sends a response to the client. In some cases, the Functional Unit sends a command to the client which returns a response. In the rest of this document, a **message** is defined as either a command or a response.

There are common commands and responses, which are commonly used across the Functional Units, and there are Functional Unit specific commands used in the Functional Unit. What kind of common commands and responses are supported, and what kind of Functional Unit specific commands are defined for the Functional Unit is described in each chapter of the Functional Unit. Mandatory support or optional support by the Functional Unit is also defined in each description of the Functional Unit. If commands are categorized mandatory support, Functional Unit must support these commands and recognize them for further handling, however support of optional commands depends on the Functional Unit implementation. The Functional Unit will inform a client what Optional support commands are supported by the supportedCommand attribute in the Capability Attribute.

The command consists of command itself and parameters to control the services in the Functional Unit. Some parameters are defined as optional, therefore Functional Unit does not need to support all of the parameters defined in the specification. The command sender can set the optional parameters if the Functional Unit supports them.

1.3.Message Sequence

A **message sequence** is the sequence of two or more related messages exchanged between the client and the Functional Unit alternately. Some message sequences are initiated by the client, and the other message sequences are initiated by the Functional Unit.

A complete message sequence consists of one command and one response which is either **ACK** or **NACK** except for data transfer message sequence which is described in "Data Transfer Messages" section on page 18.

A message sequence is initiated by a command. Every message has a parameter called MsgSeqID in its header. The MsgSeqID field is INTEGER data type. It is used as follows:

- When the client application or Functional Unit that has initiated the service session by sending an Open Service request is going to start a new message sequence, it assigns a positive value to the MsgSeqID field of the initiating command. All the rest of commands and responses in this message sequence have the same value in their MsgSeqID field.

  The MsgSeqID value is increased by one for each new message sequence. It wraps back to 1 after a sufficiently large number, e.g. 32767 (‘7FFF’ in hexadecimal).

- When the Functional Unit that has accepted the Open Service request is going to start a new message sequence, the same rule is followed except that a negative value is assigned in the MsgSeqID field.

  The MsgSeqID value is decreased by one for each new message sequence. It wraps back to -1 after a sufficiently small number, e.g. -32768 (‘8000’ in hexadecimal).

A message sequence is terminated by and only by ACK, NACK, or the closing of the service session.

When the client or the Functional Unit (FU) initiates a new message sequence, it must follow the following rules:

- The client/FU must not initiate a new message sequence while another message sequence it has initiated is not terminated.

Client	**Server**

-- start of message sequence-a --

COMMAND (MsgSeqID=1) =>

<= *COMMAND* (MsgSeqID=1)

COMMAND (MsgSeqID=1) =>

<= ACK or NACK (MsgSeqID=1)

-- end of message sequence-a --

-- start of message sequence-b --

COMMAND (MsgSeqID=2) =>

<= *COMMAND* (MsgSeqID=2)

ACK or NACK (MsgSeqID=2)=>

-- end of message sequence-b --

-- start of message sequence-c --

COMMAND (MsgSeqID=3) =>

<= ACK or NACK (MsgSeqID=3)

-- end of message sequence-c --

:

- The client/FU may initiate a new message sequence even if there is an on-going message sequence provided that the on-going message sequence has been initiated not by the client/FU but by the other end of the service session.

Client	**Server**

-- start of message sequence-a --

COMMAND (MsgSeqID=1) =>

-- start of message sequence-b --

<= *COMMAND* (MsgSeqID=-1)

ACK or NACK (MsgSeqID=-1)=>

-- end of message sequence-b --

<= ACK or NACK (MsgSeqID=1)

-- end of message sequence-a --

One whole Transfer Data packet contains one and only one message under the Salutation Personality Protocol.

1.4.Common Messages

The following messages are commonly used across Functional Units under the Salutation Personality Protocol as a part of the common framework. The specification of each Functional Unit dictates which message is actually supported and in what context it is used.

1.4.1.ACK, NACK Messages

These are the simplest forms of common response against various commands. *ACK* positively acknowledges the previous command, and *NACK* negatively acknowledges it.

ACK takes zero or more parameters, of which data type or meaning depend on the associated command. In the rest of this document, the following convention is used to describe parameters of *ACK* message.

- ACK(NULL) : ACK takes no parameter
- ACK(*type*) : ACK takes one parameter whose data type is *type*.
- ACK(*type-1, type-2*) : ACK takes two parameters. The data type of the first parameter is *type-1*, that of the second is *type-2*.
- (and so on)

NACK takes one mandatory parameter value, **ReturnCode**, that describes why the previous command is rejected or failed to be processed. *NACK* may take an optional parameter to describe the detail or additional information regarding the reason of rejection/failure. This optional parameter is mainly intended for implementation convenience (e.g. for debug or diagnostics), and may be ignored by the receiver of *NACK* response. The existence and content of this optional parameter is entirely left to each implementation.

How *ACK/NACK* is used is described in the description of each associated command.

ASN.1 Syntax Definition

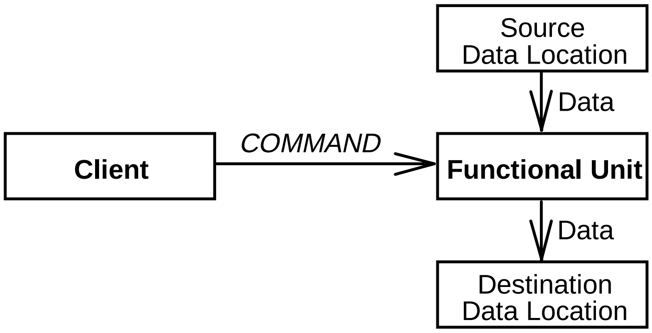
```
ACK                     ::= [APPLICATION tagACK] SEQUENCE
{
                            COMPONENTS OF MsgHeader,
    parameter1              [0] ANY OPTIONAL,
    parameter2              [1] ANY OPTIONAL,
    parameter3              [2] ANY OPTIONAL
    --  :                        :
    -- The number and data type of parameters depend on the associated command, and are defined
    -- by the specification of each associated command.
}
```

```
NACK     ::= [APPLICATION tagNACK] SEQUENCE
{
                                        COMPONENTS OF MsgHeader,
    returnCode                          [0] ReturnCode,
    descriptor                          [1] OCTET STRING        OPTIONAL
                                            -- Additional information for the reason of rejection.
                                            -- Debug/diagnostics purpose. May be ignored.
}
```

1.4.2.Data Transfer Messages

1.4.2.1.Overview

"Data" represents a meaningful unit of data sequence, such as compound document, file, etc., which is treated in a certain consistent manner by client/server applications with the Salutation Personality Protocol.



A data transfer is initiated by a *COMMAND* that is sent by a client application or another Functional Unit (the client) to a Functional Unit. Depending on the *COMMAND*, the data is transferred into the Functional Unit and/or the data is transferred out of the Functional Unit.

The *COMMAND* specifies:

- how the "data" should be processed, and
- where the "data" to be processed is (source data location) and/or where the processed "data" should be (destination data location)

Some examples of such *COMMAND* are Print (data transfer into FU), RetrieveDoc (data transfer out of FU), and commands for data format conversion (data transfer into and out of FU).

1.4.2.1.1.Data Transfer Mode

When the *COMMAND* initiates data transfer **into the Functional Unit**, the following two **Data Transfer Mode**s are defined. If the *COMMAND* is a job-request-type (refer to “Job-Related Messages” section on page 33), the *COMMAND* explicitly or implicitly specifies the mode to be used. If the *COMMAND* is not a job-request-type, data transfer is always in immediate mode.

- **Immediate Mode**

Data transfer is initiated immediately after the *COMMAND* is received by the Functional Unit (before the *COMMAND* is queued in the job queue if it is a job-request-type.)

- **Delayed Mode**

  Data transfer is initiated when the *COMMAND* is executed by the Functional Unit, after the *COMMAND* is dequeued from the job queue.

The delayed-mode data transfer is useful when the Functional Unit does not have a large storage to pool data.

1.4.2.1.2.Data Location

The source/destination data location is specified as one of the following:

- **Client**

  The source/destination data location is the client itself that is issuing the *COMMAND* to initiate the data transfer.

- **Export Pool**

  This choice may be used only as the destination data location.

  When it is specified, the “data” is not actually transferred but is prepared for another Functional Unit to access.

- **Functional Unit**

  This choice may be used only as the source data location.

  The data is transferred from the specified Functional Unit (data-source FU). The data-source FU is identified by the Functional Unit Handle of the data-source FU and the SLM-ID of the SLM with which the data-source FU is registered.

- **URL**

It is optional to support this choice. The Functional Unit indicates in its capability attribute whether or not it supports URL-based data location specification.

The data is transferred from/to a file system designated by the specified Uniform Resource Locator (URL). Only the URL with either “ftp” or “file” scheme is allowed. If the “ftp” scheme is used, the Functional Unit accesses the specified file under the File Transfer Protocol (FTP). If the “file” scheme is used, the Functional Unit accesses the specified file in an implementation dependent way.

Unlike the other three choices, if the URL-based data location is specified, it is not possible to transfer the description (format) of the “data” together with the “data” content or to retain the data block boundaries (described in “Data Transfer Message Sequence” section below). The URL-based data location should not be used when the data format is not obvious from the file name or when the data block boundaries need to be maintained.

The rest of “Data Transfer Messages” section defines the framework for the data transfer between a client application and a Functional Unit, or between a Functional Unit and another Functional Unit, and is not applicable to data transfer to/from the URL-specified data location.

1.4.2.1.3.DataHandle

DataHandle is used to identify the "data" to be transferred. DataHandle is always assigned by the sender of the data. Data-sending FU implementation must generate a sufficiently long and random value for a DataHandle.

A DataHandle is valid only for one Data Transfer Message Sequence. After the completion of the Data Transfer Message Sequence, the sender invalidates the DataHandle. If the same or different receiver attempts to request another data transfer using the same DataHandle, the sender rejects it by "Unknown DataHandle" error.

When a client application requests a Functional Unit (data-source FU) to send data to another Functional Unit (data-destination FU), the client application sets the data destination parameter in the *COMMAND* to the data-source FU as the **Export Pool**. The data-source FU generates a DataHandle and returns it to the client application. Such data handle is called **Export DataHandle**. The client application then sends another *COMMAND* with the Export DataHandle to the data-destination FU. The data-destination FU sends a *RequestDataTransfer* command (described in the next section) with the Export DataHandle to the data-source FU to initiate receiving the data.

The Export DataHandle is not invalidated even if the client application closes the service session after it receives the Export DataHandle from the data-source FU. There may be cases that the data source FU has assigned an Export DataHandle, but no receiver will have requested a data transfer with the Export DataHandle. Each implementation may define when unused Export DataHandles are invalidated, for example, once a day, or limiting the number of pending handles to twenty.

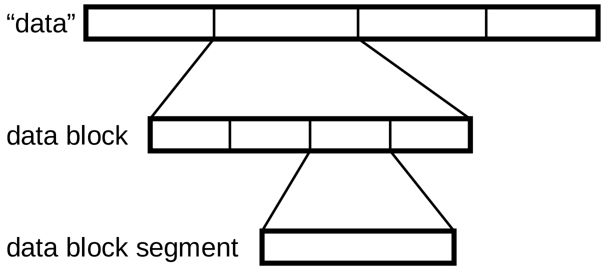
1.4.2.1.4.Data Transfer Message Sequence

A **Data Transfer Message Sequence** is used to transfer the data between the source/destination data location and the Functional Unit. It consists of the following messages:

- RequestDataTransfer
- DataBlockDescription
- TransferDataBlock
- RequestNextData
- ACK
- NACK

In this section, a "data" is defined as the whole of data to be transferred in one complete Data Transfer Message Sequence.

A "data" consists of one or more data blocks. One data block consists of one or more data block segments. One data block segment is transferred by one *TransferDataBlock* message. The message has two flags (Begin Data Block and End Data Block) to indicate data block boundaries and an additional flag (Last Segment) to indicate the last segment of the last data block.



It is allowed to mix different format (or any other optional attributes) of data blocks in a "data". (However, one data block shall not contain mixed formats). *DataBlockDescription* message is used to specify the format of subsequent data blocks. It may be omitted If the format of data block is known to the receiver, for example, because the receiver and the sender have agreed on the format by a separate message in advance, or because the specification of Functional Unit has defined the default format.

A "data" is transferred in one data block unless it contains mixed formats or the specification of each Functional Unit specifies when a "data" should be divided into multiple data blocks. For example, when a document in bi-level image stream format is printed by a [Print] Functional Unit under the Salutation Personality Protocol, the document “data” is transferred to the FU in multiple data blocks, each data block corresponding to a page in the document.

On the other hand, it is an implementation matter how to divide each data block into multiple data block segments. For example, if a particular SLM implementation imposes the maximum size for the application data parameter of *slmTransferData()* SLM-API, the sending application may have to split each data block so that each data block segment fits in the limited size.

The following is the simplest form of Data Transfer Message Sequence.

Sender	**Receiver**
TransferDataBlock(Begin, End, Last) =>	
	<= ACK

The following is more complex example of Data Transfer Message Sequence. In this example, the "data" consists of three data blocks, the first two in one format, and the third in another format. The second data block is further divided into two data block segments.

Sender		**Receiver**
		<= RequestDataTransfer(DataHandle)
	data format for the first & second data block	
DataBlockDescription =>		
		<= RequestNextData
	the first data block	
TransferDataBlock(Begin, End) =>		
		<= RequestNextData
	the second data block	
TransferDataBlock(Begin) =>		
		<= RequestNextData
TransferDataBlock(End) =>		
		<= RequestNextData
	data format for the third data block	
DataBlockDescription =>		
		<= RequestNextData
	the third data block	
TransferDataBlock(Begin, End, Last) =>		
		<= ACK

The same MsgSeqID value is used in all messages in a Data Transfer Message Sequence.

1.4.2.1.5.Simplified Data Transfer Message Sequence

When a *COMMAND* initiates the transfer of **only one "data" from the client** to the Functional Unit under **immediate mode**, the Functional Unit omits the ACK response to the *COMMAND* and immediately sends a RequestDataTransfer message.

When a *COMMAND* initiates the transfer of **only one "data"** from the Functional Unit **to the client**, the Functional Unit omits both ACK response and RequestDataTransfer message.

In both cases, DataHandle is not used.

The same MsgSeqID value is used in all messages from the initiating *COMMAND* through the completion of the data transfer in a simplified data transfer message sequence.

1.4.2.1.6.Usage of Data Transfer Message Sequence

The following examples show typical uses of Data Transfer Message Sequence. In the examples, the shortest possible Data Transfer Message Sequence is shown only for the sake of simplicity, however the "data" may actually consist of multiple data blocks, each data block consisting of multiple data block segments.

1.4.2.1.6.1.Data Transfer from Client to Functional Unit : Delayed Mode

The client issues a *COMMAND* with a DataHandle that identifies the client's data to be transferred to the Functional Unit. The *COMMAND* may be queued and not processed immediately by the Functional Unit. The Functional Unit requests the client to transfer the data when the *COMMAND* is dequeued and executed.

Client	**Server**
--- Message Sequence Start (MsgSeqID=n1)---	
COMMAND(DataHandle, ...) =>	
	<= ACK(...)
--- Message Sequence End (MsgSeqID=n1)---	
COMMAND may be queued.	
--- Message Sequence Start (MsgSeqID=n2)---	
	<= RequestDataTransfer(DataHandle)
TransferDataBlock(Begin, End, Last) =>	
	<= ACK(NULL)
--- Message Sequence End (MsgSeqID=n2)---	

If more than one data item is to be transferred, the Functional Unit initiates a separate Data Transfer Message Sequence for each data item, as follows.

Client	**Server**

--- Message Sequence Start (MsgSeqID=n1)---

COMMAND(DataHandle-1, DataHandle-2, ...) =>

<= ACK(...)

--- Message Sequence End (MsgSeqID=n1)---

COMMAND may be queued.

--- Message Sequence Start (MsgSeqID=n2)---

<= RequestDataTransfer(DataHandle-1)

TransferDataBlock(Begin, End, Last) =>

<= ACK(NULL)

--- Message Sequence End (MsgSeqID=n2)---

--- Message Sequence Start (MsgSeqID=n3)---

<= RequestDataTransfer(DataHandle-2)

TransferDataBlock(Begin, End, Last) =>

<= ACK(NULL)

--- Message Sequence End (MsgSeqID=n3)---

1.4.2.1.6.2.Data Transfer from Client to Functional Unit : Immediate Mode

If more than one data item is to be transferred, the flow is the same as above except that data items are transferred before the *COMMAND* is queued.

If only one data item is to be transferred, simplified data transfer message sequence is used as follows:

Client	**Server**

--- Message Sequence Start (MsgSeqID=n1)---

COMMAND(...) =>

<= RequestDataTransfer()

TransferDataBlock(Begin, End, Last) =>

<= ACK(...)

--- Message Sequence End (MsgSeqID=n1)---

1.4.2.1.6.3.Data Transfer from Functional Unit to Client

If more than one data item is to be transferred, the client application initiates a separate Data Transfer Message Sequence for each data item, as follows.

Client	Server
--- Message Sequence Start (MsgSeqID=n1)---	
COMMAND(...) =>	
	<= ACK(DataHandle-1, DataHandle-2, ...)
--- Message Sequence End (MsgSeqID=n1)---	
--- Message Sequence Start (MsgSeqID=n2)---	
RequestDataTransfer(DataHandle-1) =>	
	<= TransferDataBlock(Begin, End, Last)
ACK(NULL) =>	
--- Message Sequence End (MsgSeqID=n2)---	
--- Message Sequence Start (MsgSeqID=n3)---	
RequestDataTransfer(DataHandle-2) =>	
	<= TransferDataBlock(Begin, End, Last)
ACK(NULL) =>	
--- Message Sequence End (MsgSeqID=n3)---	

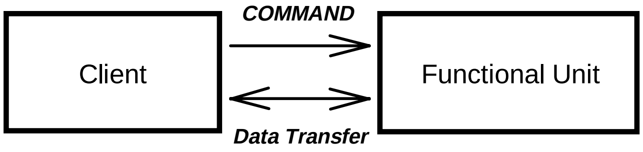
If only one data item is to be transferred, simplified data transfer message sequence is used as follows:

Client	Server
--- Message Sequence Start (MsgSeqID=n1)---	
COMMAND(...) =>	
	<= TransferDataBlock(Begin, End, Last)
ACK(NULL) =>	
--- Message Sequence End (MsgSeqID=n1)---	

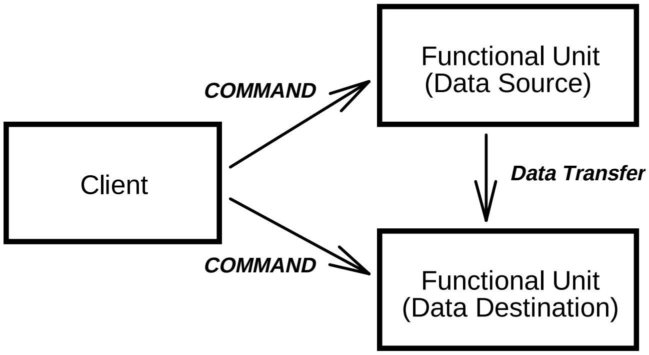
1.4.2.1.6.4.Data Transfer between Functional Units

Examples so far have shown data transfers between the client and the Functional Unit. A client application can also request a data transfer to occur between two Functional Units.

Data Transfer between Client and Functional Unit



Data Transfer between Functional Units



The procedure for a data transfer between Functional Units is as follows:

1) The client establishes a service session with the data-source Functional Unit.

2) The client sends a *COMMAND* to the data-source FU specifying that the destination of data transfer is "Export Pool" (not the client). When the Export Pool is selected as the destination, the data is not actually transferred but is just prepared for another Functional Unit to access. The data-source FU assigns an Export DataHandle for the data item to be transferred, and returns it to the client. (The session between the client and the data-source FU may be closed hereafter.)

3) The client establishes a service session with the data-destination Functional Unit.

4) The client sends a *COMMAND* to the data-destination FU to give the Export DataHandle together with **AbsoluteFunctionalUnitHandle**. AbsoluteFunctionalUnitHandle contains the following information of the data-source FU:

   - The SLM-ID of the SLM with which the data-source FU is registered
   - The Functional Unit Handle of the data-source FU

   (The session between the client and the data-destination FU may be closed hereafter.)

5) The data-destination Functional Unit establishes a service session with the data-source Functional Unit.

6) The data-destination FU initiates the data transfer message sequence by sending a RequestDataTransfer message with the Export DataHandle.

7) The data-destination Functional Unit closes the session with the data-source Functional Unit after the data transfer is completed.

The following figure shows how messages are actually exchanged.

Client	**Data-Source FU**

Client establishes a session with data-source Functional Unit.

--- Message Sequence Start (MsgSeqID=n1)---

COMMAND(DataDestination=ExportPool, ...) =>

<= ACK(DataHandle)

--- Message Sequence End (MsgSeqID=n1)---

Client	**Data-Destination FU**

Client establishes a session with data-destination Functional Unit.

--- Message Sequence Start (MsgSeqID=n2)---

COMMAND(DataSource=AbsoluteFunctionalUnitHandle, DataHandle, ...) =>

<= ACK(...)

--- Message Sequence End (MsgSeqID=n2)---

Data-Destination FU	**Data-Source FU**

Session is established between the Functional Units.

--- Message Sequence Start (MsgSeqID=n3)---

RequestDataTransfer(DataHandle) =>

<= TransferDataBlock(Begin, End, Last)

ACK(NULL) =>

--- Message Sequence End (MsgSeqID=n3)---

1.4.2.1.7.Skeleton of Data-Transfer-Initiating Command

Some of the parameters shown in the following skeleton may be defined as optional or not included in each data-transfer-initiating *COMMAND* definition.

```
COMMAND                    ::= [APPLICATION tagCOMMAND] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
      : (other parameters)
   modeOfDataTransfer           [ ] DataTransferMode,
   dataSource                   [ ] DataLocation          DEFAULT client,
   dataHandle                   [ ] DataHandle,
   dataDestination              [ ] DataLocation          DEFAULT client,
      : (other parameters)
}
```

1.4.2.2.Message Description

1.4.2.2.1.RequestDataTransfer

The receiver of the data initiates the Data Transfer Message Sequence by this message. It identifies the "data" to be transferred by **DataHandle** parameter. DataHandle must have been assigned by the sender and known to the receiver by a separate message in advance, as described in the previous "Usage of Data Transfer Message Sequence" section on page 22.

The DataHandle parameter in RequestDataTransfer message is omitted in simplified data transfer message sequence.

Response

One of the following is returned in response to this message:

- *DataBlockDescription*, *TransferDataBlock*

  The sender has accepted the request and started the data transfer.

- *NACK*

  The sender has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure. *NACK* terminates the message sequence.

ASN.1 Syntax Definition

```
RequestDataTransfer        ::= [APPLICATION tagRequestDataTransfer] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   dataHandle                   [0] DataHandle OPTIONAL
}
```

1.4.2.2.2.DataBlockDescription

This message tells the receiver of the format and/or other attributes of subsequent data blocks in this Data Transfer Message Sequence.

The specification of each Functional Unit dictates whether this message is used or not, when this message is used, and what parameter this message takes.

Response

One of the following is returned in response to this message:

- *RequestNextData*

  The receiver has processed the message successfully and is ready to receive next message.

- *NACK*

  The receiver has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure. *NACK* terminates the message sequence.

ASN.1 Syntax Definition

```
DataBlockDescription            ::= [APPLICATION tagDataBlockDescription] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    dataDescriptor                  [0] CHOICE
    {
        document                    [0] DocumentDataDescriptor,
        file                        [1] FileData
    }
}
```

1.4.2.2.3.TransferDataBlock

This message is used by the sender to transfer a data block segment to the receiver. A separate *TransferDataBlock* message is used for each data block segment.

It has flags to indicate the first and last data block segment in a data block. If a data block consists of only one data block segment, both flags are set and entire data block is transferred by one *TransferDataBlock* message.

This message also has another flag to indicate the last data block segment of the last data block. For example, if the entire “data” is transferred by one *TransferDataBlock* message, all the three flags are set.

Note: “TransferDataBlock” message should not be confused with “*Transfer Data*” RPC message of the Salutation Manager Protocol.

Response

One of the following is returned in response to this message:

- *RequestNextData*

  The receiver has processed the message successfully and is ready to receive next message.

- *ACK*

  This response is returned only if the "last data block segment" flag is set in the preceding *TransferDataBlock* message, and the receiver has successfully processed the message. This is the last message of a successful Data Transfer Message Sequence.

  Unless otherwise specified by the specification of each Functional Unit, *ACK* takes no parameter.

- *NACK*

  The receiver has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure. *NACK* terminates the message sequence.

ASN.1 Syntax Definition

```
TransferDataBlock          ::= [APPLICATION tagTransferDataBlock] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    beginDataBlock               [0] BOOLEAN,
    endDataBlock                 [1] BOOLEAN,
    lastSegment                  [2] BOOLEAN,  -- TRUE in the last data block segment of the
                                               -- last data block of “data”
    dataBlockBody                [3] OCTET STRING
}
```

1.4.2.2.4.RequestNextData

The receiver issues this message when it has successfully processed the previous message from the sender, and when it is ready to receive next message.

Response

One of the following is returned in response to this message:

- *DataBlockDescription*, *TransferDataBlock*

  The sender is continuing the data transfer.

- *NACK*

  The sender has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure. *NACK* terminates the message sequence.

ASN.1 Syntax Definition

```
RequestNextData            ::= [APPLICATION tagRequestNextData] SEQUENCE
{
                                 COMPONENTS OF MsgHeader
}
```

1.4.3.Attribute Repository Messages

1.4.3.1.Overview

The following messages are defined to read/set values of attributes in the Attribute Repository. Each message may include more than one attribute.

- GetPrivateAttribute
- GetGlobalAttribute
- SetPrivateAttribute

All of these messages are not mandatory support for the Functional Unit. Supported message tags by an FU will be set in the supportedCommand attribute in the capability attributes.

The normal flow of these commands and responses are as follows:

Client	Server
To read attribute values:	
GetPrivateAttribute(SET OF AttributeID) => or GetGlobalAttribute(SET OF AttributeID) =>	
	<= ACK(AttributeList)
To set attribute values:	
SetPrivateAttribute(AttributeList) =>	
	<= ACK(NULL)

The initial version of the architecture has not defined SetGlobalAttribute message.

1.4.3.2.Message Description

1.4.3.2.1.GetPrivateAttribute, GetGlobalAttribute

The client sends a *GetPrivateAttribute* or a *GetGlobalAttribute* message to read the value of one or more Private Attributes or Global Attributes respectively. The message has a set of Attribute IDs as its parameter.

The server returns the attribute ID-value pairs of only those attributes that exist in the Attribute Repository. The attributes that do not exist, either because they are not supported or because their values have not been set by SetPrivateAttribute message, are not included in the response.

Response

One of the following is returned in response to this message:

- *ACK*(AttributeList)

  The server is returning the valid ID-value pairs of queried attributes.

- *NACK*(ReturnCode)

  The server has failed to process the command. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
GetPrivateAttribute          ::= [APPLICATION tagGetPrivateAttribute] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    attributeIdList              [0] SET OF AttributeID
}
```

```
GetGlobalAttribute              ::= [APPLICATION tagGetGlobalAttribute] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    attributeIdList                 [0] SET OF AttributeID
}

AttributeList                   ::= SET OF SEQUENCE
{
    attributeId                     [0] AttributeID,
    attributeValue                  [1] ANY          -- Type is defined by each attribute.
}
```

1.4.3.2.2.SetPrivateAttribute

The client sends a *SetPrivateAttribute* message to set the value of one or more Private Attributes respectively. The message has a set of Attribute ID-value pairs as its parameter.

If any of the attributes in the message is unknown or not supported by the server, the entire message is rejected and no attribute value is changed.

If the value parameter is omitted, the corresponding attribute in the Attribute Repository is deleted. The command is accepted even if the attribute to be removed does not exist in the Attribute Repository provided that the attribute is supported.

Response

One of the following is returned in response to this message:

- *ACK*(NULL)

  The server has successfully set attribute values.

- *NACK*(ReturnCode)

  The server has failed to set attribute values. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
SetPrivateAttribute             ::= [APPLICATION tagSetPrivateAttribute] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    attributeList                   [0] SET OF SEQUENCE
    {
        attributeId                 [0] AttributeID,
        attributeValue              [1] ANY          OPTIONAL
                                        -- Type is defined by each attribute.
                                        -- If omitted, attribute is deleted.
    }
}
```

1.4.4.Job-Related Messages

1.4.4.1.Overview

1.4.4.1.1.Job-Request-Type Command and Job Entry

For a command that may take time to be processed, the server assigns and returns a **JobHandle** to the client in *ACK* response as follows, when the command is received but before the command is executed.

Client	**Server**
COMMAND =>	
	<= ACK(JobHandle)
	(The COMMAND is enqueued in the job queue.)
	:
	(The COMMAND is dequeued from the job queue and executed.)

This type of command is called **Job-Request-Type** command. The client may send a *CancelJob* message to the server to cancel the associated job-request-type command. The server deletes the command if it is in a queue and not processed yet, or optionally abort the command execution if it is being processed. See the description of *CancelJob* message for more detail.

Client	**Server**
CancelJob(JobHandle) =>	
	<= ACK(NULL)

Certain job-request-type commands are structured such that a job consists of more than one task. For example, a command to send a document to multiple destinations by FAX consists of multiple tasks. Each task sends a document to one named destination. Each task in a job-request-type command is called a **Job Entry**. The client assigns a unique integer value, called **JobEntryID**, to each Job Entry. Note that JobHandle is assigned by the server but JobEntryID is assigned by the client. JobEntryID can be used, for example, to know the result of each Job Entry execution individually later.

Job or Job Entry related messages are not mandatory support for the Functional Unit. Supported message tags by an FU will be set in supportedCommand attribute in the capability attributes.

1.4.4.1.2.Life of Job

JobHandle, JobEntryID, and the status of job execution have one of the following life. The server must retain these values, and the client may refer to JobHandle and JobEntryID in messages, only during their life.

- Job (default)

  The life terminates after the server sends the result of job execution to the client at the completion of the job, or if there is no notification to be sent, the life terminates at the completion of job execution. The server must retain JobEntryID and status of all job entries until all job entries are executed.

Note that the life is NOT terminated by the termination of current Service Session (*Open Service ~ Close Service*).

- Session

  The life terminates at the end of current Service Session (*Open Service ~ Close Service*). It may be terminated also by *FreeJobHandle* message from the client.

  If the job is still queued in the job queue or it is being executed when the current session is terminated, the job is purged from the job queue (together with any spooled data, if any) or the job execution is aborted respectively.

  The server is desirable to retain JobEntryID and status of all job entries until current session is terminated or JobHandle is freed.

- Persistent

  The life is not terminated by the end of current Service Session or by the completion of the job execution. The life is terminated either:

  - □ by *FreeJobHandle* message from the client, or
  - □ by any implementation defined way. For example, a Functional Unit may be implemented such that it retains only the last twenty job results, a new one always replacing the oldest one.

  The server is desirable to retain JobEntryID and status of all job entries persistent.

The Life parameter is included in the job-request-type command to explicitly specify the life of JobHandle, JobEntryID, and job status. If the Life parameter is omitted, the life of "Job" is assumed.

In any case, the life is terminated if the job is canceled by the client before or during job execution.

1.4.4.1.3.Job Status Notification

A job or job entry is in one of the following state:

- Queued : execution has not begun
- Started : execution is in progress
- Suspended : execution is temporarily suspended
- Completed : execution has successfully completed
- Error : execution has completed in error
- WaitingForScheduledTime : waiting for the scheduled execution start time
- Canceled : queued job or job entry is canceled
- Aborted : execution is aborted

Following figure shows the relationship among job status and how job status is transited to another status.

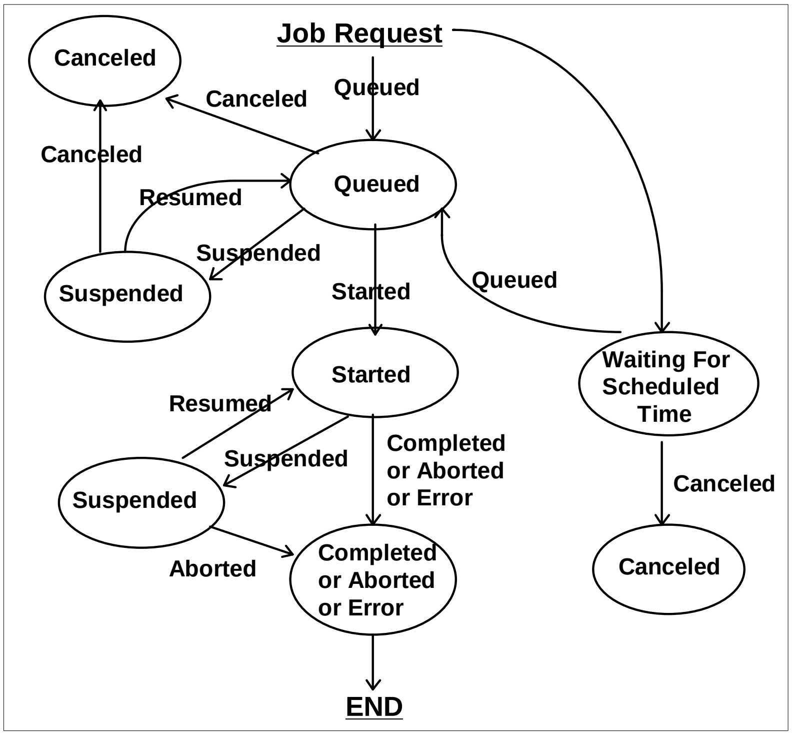


Fig 1 Job status transition table

Note

JobStatusCode includes both job status and the transaction. Only NotifyJobStatus uses the transaction part and the others, e.g. JobStatusNotificationMode/FUJobList, use the job status part. Above figure shows the job-status transaction table, a job status name in an oval and a transaction name along an arrow.

If the job consists of multiple job entries, how job entry status transition will affect to the job status will be described in each Functional Unit part.

The client can either solicit the server to know the current status of a job or job entries, or request the server to notify the client at certain status changes of a job or job entries.

When the client solicits the server for the status of a job, it sends a QueryJobStatus message to the server.

Client	Server
QueryJobStatus(JobHandle) =>	
	<= ACK(JobStatusCode, ...)

When the client solicits the server for the status of a specific job entry or all the job entries of a job, it sends a QueryJobEntryStatus message to the server.

Client	Server
QueryJobEntryStatus(JobHandle, JobEntryID) =>	
	<= ACK(JobStatusCode, ...)

When the client requests the server to notify certain status changes of a job or the job entries of a job:

- a **JobStatusNotificationMode** parameter is included in the job-request-type command, or
- the client sends StartMonitorJobStatus command which includes a JobStatusNotificationMode parameter. (The clients sends CancelMonitorJobStatus command when it no longer needs to be notified of job status changes.)

The JobStatusNotificationMode parameter specifies:

- what types of status changes should be notified, and
- whether the notification is requested for the job on the whole, or the notification is requested for each job entry of the job.

When the JobStatusNotificationMode indicates the notification is for the job on the whole:

Client	Server

COMMAND(..., JobStatusNotificationMode, ...) =>

<= ACK(JobHandle)

:

A notification is sent when the job execution is

- *started,*
- *suspended,*
- *resumed,*
- *completed successfully,*
- *completed in error,*
- *canceled, or*
- *aborted*

as requested:

<= NotifyJobStatus(JobHandle, JobStatusCode, ...)

ACK(NULL) =>

:

<= NotifyJobStatus(JobHandle, JobStatusCode, ...)

ACK(NULL) =>

:

When the JobStatusNotificationMode indicates the notification is for each job entry:

Client **Server**

COMMAND(..., JobStatusNotificationMode, ...) =>

<= ACK(JobHandle)

:

A notification is sent when each job entry execution is

- *started,*
- *suspended,*
- *resumed,*
- *completed successfully,*
- *completed in error,*
- *canceled, or*
- *aborted*

as requested:

<= NotifyJobEntryStatus(JobHandle, JobEntryID, JobStatusCode, ...)

ACK(NULL) =>

<= NotifyJobEntryStatus(JobHandle, JobEntryID, JobStatusCode, ...)

ACK(NULL) =>

:

<= NotifyJobEntryStatus(JobHandle, JobEntryID, JobStatusCode, ...)

ACK(NULL) =>

<= NotifyJobEntryStatus(JobHandle, JobEntryID, JobStatusCode, ...)

ACK(NULL) =>

:

If the JobStatusNotificationMode parameter is not included in the job-request-type command, no notification is made unless the client issues a StartMonitorJobStatus command.

If the life of “job” or “persistent” is specified in the job-request-type command, the client application may close the service session after it receives the ACK for the command even if the job execution has not completed (or even begun). If any job status notifications are requested, the client application that is to receive the notifications must have registered itself with the Salutation Manager (SLM) as a [Client] Functional Unit.

By default, the job status notification is sent to the client application that submitted the job. Because the Functional Unit Handle of the job-requesting [Client] FU and the SLM-ID of the SLM with which the [Client] FU is registered are passed to the server FU at the *Open Service* request, the server FU will be able to send an *Open Service* request to the [Client] FU to send job status notifications if no service session exists.

It is possible for the job-requesting client application to specify that the job status notifications are sent to another [Client] FU. The notification-receiving application must have registered itself as a [Client] FU. The job-requesting client application specifies the Functional Unit Handle of the [Client] FU and the SLM-ID of the SLM with which the [Client] FU is registered in the job-requesting command as “NotificationScheme” parameter.

If the "check interval" parameter is specified in the job-request-type command, the Functional Unit has to request the Salutation Manager (SLM), by calling slmStartAvailabilityCheck(), to periodically check if the [Client] FU to receive the job status notifications is still available. If this parameter is not specified, no Availability Check is performed.

1.4.4.1.4.Job Control Attribute

Job-request-type commands sometimes have parameters called "**Job Control Attributes**". An example of a job control attribute is "Job Priority" attribute.

A particular job control attribute is associated with either the whole job or a job entry.

The following commands may be used to change the value of a job control attribute of a queued job:

- ChangeJobAttribute
- ChangeJobEntryAttribute

The specification of each command dictates which job control attributes support these commands.

1.4.4.1.5.Job Suspend/Resume

The following commands are defined to suspend the execution of queued or currently executing job or job entry:

- SuspendJob
- SuspendJobEntry

The following commands are defined to resume the suspended job or job entry:

- ResumeJob
- ResumeJobEntry

The specification of each job-request-type command dictates whether these commands are supported or not.

1.4.4.1.6.List FU Job Status

List FU Job Status type command is defined for the client to get the list of current job status in the Functional Unit. The supported command for each Functional Unit is defined in each chapter.

The example flow of the command and its response is as follows:

Client	**Server**
ListFUJob(...) =>	
	<= TransferDataBlock(FUJobList)
ACK(NULL) =>	

1.4.4.1.7.Skeleton of Job-Request-Type Command

Not all the parameters shown in the following skeleton of the job-request-type command are defined in each job-request-type command.

```
COMMAND                       ::= [APPLICATION tagCOMMAND] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
      : (other parameters)
   life                            [ ] Life                     DEFAULT job,
   jobStatusNotificationMode       [ ] JobStatusNotificationMode    OPTIONAL,
                                      -- If omitted, no notification is made.
   notificationScheme              [ ] NotificationScheme       OPTIONAL,
                                      -- Omitted unless the job status notifications are to be
                                      -- sent to a [Client] FU other than the client that is
                                      -- sending this command
   checkInterval                   [ ] INTEGER                  OPTIONAL
                                      -- Interval (in seconds) for the FU-side SLM to periodically
                                      -- check the availability of the [Client] FU to receive the job
                                      -- status notification
                                      -- If omitted, the Availability Check is not performed.
   jobControlAttributes            [ ] ...
      : (other parameters)
}
```

1.4.4.2.Message Description

1.4.4.2.1.QueryJobStatus

The client sends this message to the server to query the status of a job.

It may be sent after the client receives a JobHandle in the response to a job-request-type command the client has issued, and before the life of JobHandle, JobEntryID, and job status is terminated.

Response

One of the following is returned in response to this message:

- *ACK*(JobStatusCode, ReasonCode)

  JobStatusCode indicates the status of the job. If JobStatusCode is either "error" or "suspended", ReasonCode is included and indicates the reason of the suspension or error.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
QueryJobStatus              ::= [APPLICATION tagQueryJobStatus] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    jobHandle                   [0] JobHandle
}
```

1.4.4.2.2.QueryJobEntryStatus

The client sends this message to the server to query the status of a specific job entry or all the job entries of a job.

It may be sent after the client receives a JobHandle in the response to a job-request-type command the client has issued, and before the life of JobHandle, JobEntryID, and job status is terminated.

Response

One of the following is returned in response to this message:

- *ACK*(JobStatusCode, ReasonCode) or *ACK*(JobEntriesStatus)

  If the status of a specific job entry is queried, *ACK* includes JobStatusCode and optional ReasonCode. JobStatusCode indicates the status of the job entry. If JobStatusCode is either "error" or "suspended", ReasonCode is included and indicates the reason of the suspension or error.

  If the status of all the job entries is queried, *ACK* includes JobEntriesStatus.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
QueryJobEntryStatus         ::= [APPLICATION tagQueryJobEntryStatus] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    jobHandle                   [0] JobHandle,
    jobEntryId                  [1] JobEntryID    OPTIONAL
                                    -- If omitted, the status of all the job entries is requested.
}
```

```
JobEntriesStatus                 ::= SET OF SEQUENCE
{
    jobEntryId                       [0] JobEntryID,
    jobEntryStatusCode               [1] JobStatusCode,
    jobEntryReasonCode               [2] ReasonCode            OPTIONAL
                                         -- present only if jobEntryStatusCode=suspended or error
}
```

1.4.4.2.3.NotifyJobStatus

The server sends *NotifyJobStatus* to the client to tell the job status change such as the completion of the job execution. The client has to indicate which types of job status changes it wishes to be notified by the JobStatusNotificationMode parameter of job-request-type command.

Response

One of the following is returned by the client in response to *NotifyJobStatus*:

- *ACK*(NULL)

  The client has received the notification successfully.

- *NACK*(ReturnCode)

  The client has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
NotifyJobStatus                  ::= [APPLICATION tagNotifyJobStatus] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
    jobHandle                        [0] JobHandle,
    jobStatusCode                    [1] JobStatusCode,
    reasonCode                       [2] ReasonCode            OPTIONAL
                                         -- present only if jobStatusCode=suspended or error
}
```

1.4.4.2.4.NotifyJobEntryStatus

The server sends *NotifyJobEntryStatus* to the client to tell the job entry status change such as the completion of the job entry execution. The client has to indicate which types of job entry status changes it wishes to be notified by the job status notification mode parameter of job-request-type command.

Response

One of the following is returned by the client in response to *NotifyJobEntryStatus*:

- *ACK*(NULL)

  The client has received the notification successfully.

- *NACK*(ReturnCode)

  The client has failed to process the *NotifyJobStatus*. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
NotifyJobEntryStatus           ::= [APPLICATION tagNotifyJobEntryStatus] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
   jobHandle                       [0] JobHandle,
   jobEntryId                      [1] JobEntryID,
   jobStatusCode                   [2] JobStatusCode,
   reasonCode                      [3] ReasonCode           OPTIONAL
                                       -- present only if jobStatusCode=suspended or error
}
```

1.4.4.2.5.ChangeJobAttribute

The client sends this message to change the value of a job control attribute parameter of a queued job-request-type command.

If the execution of the specified job has already been completed, the command is rejected by NACK (ReturnCode = rcJobAlreadyExecuted) response. (ReturnCode = rcInvalidJobHandle is used if the life of JobHandle has already expired.)

If the specified job is being executed, it is implementation dependent whether:

- the command is rejected by NACK (ReturnCode = rcJobAlreadyExecuted) response, or
- the command is accepted and the job is executed under the updated attribute value.

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has updated the job control attribute value successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
ChangeJobAttribute             ::= [APPLICATION tagChangeJobAttribute] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
   jobHandle                       [0] JobHandle,
   attributeId                     [1] AttributeID,
   attributeValue                  [2] ANY
}
```

1.4.4.2.6.ChangeJobEntryAttribute

The client sends this message to change the value of a job control attribute parameter of a job entry of a queued job-request-type command.

If the execution of the specified job entry has already been completed, the command is rejected by NACK (ReturnCode = rcJobAlreadyExecuted) response. (ReturnCode = rcInvalidJobHandle is used if the life of JobHandle has already expired.)

If the specified job entry is being executed or if the specified job is being executed but the execution of the specified job entry has not begun, it is implementation dependent whether:

- the command is rejected by NACK (ReturnCode = rcJobAlreadyExecuted) response, or
- the command is accepted and the job entry is executed under the updated attribute value.

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has updated the job control attribute value successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
ChangeJobEntryAttribute         ::= [APPLICATION tagChangeJobEntryAttribute] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    jobHandle                       [0] JobHandle,
    jobEntryId                      [1] JobEntryID,
    attributeId                     [2] AttributeID,
    attributeValue                  [3] ANY
}
```

1.4.4.2.7.SuspendJob

The client sends this message to suspend the execution of a queued or currently executing job-request-type command.

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has suspended the job successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
SuspendJob                    ::= [APPLICATION tagSuspendJob] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   jobHandle                      [0] JobHandle
}
```

1.4.4.2.8.SuspendJobEntry

The client sends this message to suspend the execution of a queued or currently executing job entry of a job-request-type command.

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has suspended the job entry successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
SuspendJobEntry               ::= [APPLICATION tagSuspendJobEntry] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   jobHandle                      [0] JobHandle,
   jobEntryId                     [1] JobEntryID
}
```

1.4.4.2.9.ResumeJob

The client sends this message to resume a suspended job-request-type command.

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has processed the message successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
ResumeJob                     ::= [APPLICATION tagResumeJob] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   jobHandle                      [0] JobHandle
}
```

1.4.4.2.10.ResumeJobEntry

The client sends this message to resume a suspended job entry of a job-request-type command.

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has processed the message successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
ResumeJobEntry                   ::= [APPLICATION tagResumeJobEntry] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
    jobHandle                        [0] JobHandle,
    jobEntryId                       [1] JobEntryID
}
```

1.4.4.2.11.CancelJob

The client sends this message to cancel a job-request-command it has issued before.

It may be sent after the client receives a JobHandle in the response to a job-request-type command the client has issued, and before the life of JobHandle is terminated.

This message includes "abort" flag. If the flag is set, the job is aborted even if the job is being executed. If the flag is NOT set and the server has already started the execution of the job, *CancelJob* message is rejected by NACK (ReturnCode = rcJobAlreadyExecuted) response.

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has canceled the job successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
CancelJob                   ::= [APPLICATION tagCancelJob] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    jobHandle                   [0] JobHandle,
    abort                       [1] BOOLEAN
                                    -- if TRUE, job is canceled either queued or being executed
                                    -- if FALSE, job is canceled only if execution has not started
}
```

1.4.4.2.12.CancelJobEntry

The client sends this message to cancel a job entry in a job-request-command it has issued before.

It may be sent after the client receives a JobHandle in the response to a job-request-type command the client has issued, and before the life of JobHandle is terminated.

This message includes "abort" flag. If the flag is set, the job entry is aborted even if the job entry is being executed. If the flag is NOT set and the server has already started the execution of the job entry, *CancelJobEntry* message is rejected by NACK (ReturnCode = rcJobAlreadyExecuted) response. The support of this feature is optional. If the server implementation does not support it, *CancelJobEntry* message with abort=TRUE received while the job entry is being executed is rejected by NACK (ReturnCode = rcJobAlreadyExecuted).

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has canceled the job entry successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
CancelJobEntry              ::= [APPLICATION tagCancelJobEntry] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    jobHandle                   [0] JobHandle,
    jobEntryId                  [1] JobEntryID,
    abort                       [2] BOOLEAN
                                    -- if TRUE, job is canceled either queued or being executed
                                    -- if FALSE, job is canceled only if execution has not started
}
```

1.4.4.2.13.FreeJobHandle

The client sends this message to tell the server that the JobHandle, JobEntryID, and job result no longer need to be retained.

It may be sent after the client receives a JobHandle in the response to a job-request-type command the client has issued, and before the life of JobHandle is terminated.

However, if the specified job execution has not been started or completed either successfully or unsuccessfully, the command is rejected by NACK (ReturnCode = rcJobNotCompleted).

In practice, this message is used by the client after the client first solicits the result of job execution by QueryJob(Entry)Status message when the life of job is "Persistent".

Response

One of the following is returned by the server in response to this message:

- *ACK*(NULL)

  The server has processed the command successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
FreeJobHandle                ::= [APPLICATION tagFreeJobHandle] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    jobHandle                    [0] JobHandle
}
```

1.4.4.2.14.StartMonitorJobStatus

The client sends *StartMonitorJobStatus* to the server to be notified of the job status change such as the completion of the job execution. The client has to indicate which types of job status changes it wishes to be notified with JobStatusNotificationMode parameter. The StartMonitorJobStatus resets the FU's jobStatusNotificationMode. For example, the StartMonitorJobStatus's JobStatusNotificationMode overrides the job-request-type command's jobStatusNotificationMode if any.

Response

One of the following is returned by the server in response to *StartMonitorJobStatus* :

- *ACK*(NULL)

  The server has received the command successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
StartMonitorJobStatus           ::= [APPLICATION tagStartMonitorJobStatus] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    jobHandle                       [0] JobHandle,
    jobStatusNotificationMode       [1] JobStatusNotificationMode
}
```

1.4.4.2.15.CancelMonitorJobStatus

The client sends *CancelMonitorJobStatus* to the server to cancel the job-status-monitoring.

Response

One of the following is returned by the server in response to *CancelMonitorJobStatus* :

- *ACK*(NULL)

  The server has received the command successfully.

- *NACK*(ReturnCode)

  The server has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
CancelMonitorJobStatus          ::= [APPLICATION tagCancelMonitorJobStatus] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    jobHandle                       [0] JobHandle
}
```

1.4.4.2.16.List FU Job Status type command

The client sends a *List FU Job Status type* message to get the list of current job in the Functional Unit. The list is transferred from the Functional Unit to the client by using a Data Transfer message Sequence.

The list of job is transferred as ‘data’ consisting of one data block which may be split into multiple data block segments. Each data block segment is FUJobDescription (The format of each FUJobDescription is defined in each FU.) which is defined as SET OF these Descriptions. For example, if Functional Unit has 900 jobs, the Description of the first 300 jobs may be sent in the first data block segment, next 300 jobs in the second data block segment, and last 300 jobs in the last data block segment. Although each data block segment contains only a part of the whole folder set, the receiving application can decode each data block segment without waiting for the next data block segment.

If the specified parameter is unknown or not supported by the FU, the command is rejected by NACK.

Response

One of the following is returned in response to this message:

- *ACK*(ANY)

The client has successfully processed the received data.

- *NACK*(ReturnCode)

  The server has failed to process the command. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

Refer to each Functional Unit description.

1.4.5.Dynamic Status Messages

1.4.5.1.Overview

A **Dynamic Status Parameter** describes an aspect of the current Functional Unit status. There are three ways for the client to access dynamic status parameters:

- Query Dynamic Status

  The client may query the current value of a particular dynamic status parameter.

- Event Notification

  The client may request the Functional Unit to notify the client of any changes of the value of a particular dynamic status parameter.

- List Job Status

  The client may request the Functional Unit to get the list current job status.

The supported Dynamic Status ID is defined in the Capability Attribute.

1.4.5.1.1.Query Dynamic Status

QueryDynamicStatus command is defined for the client to get the current value of a particular Dynamic Status Parameter which describes an aspect of the Functional Unit status.

The flow of the command and its response is as follows:

Client	**Server**
when successful:	
QueryDynamicStatus(DynamicStatusID) =>	
	<= ACK(*value*)
when unsuccessful:	
QueryDynamicStatus(DynamicStatusID) =>	
	<= NACK(ReturnCode)

1.4.5.1.2.Event Notification

The client first subscribes to one or more dynamic status parameters by sending a SubscribeEvent command to the Functional Unit.

When the value of a subscribed dynamic status parameter changes, the Functional Unit notifies the client of the new value of the dynamic status parameter by sending a NotifyEvent message.

The client sends an UnsubscribeEvent command to the Functional Unit if it no longer wishes to be notified of the changes of the dynamic status parameter value(s).

The normal flow of messages is as follows:

Client	Server
first, client subscribes to dynamic status parameter(s):	
SubscribeEvent(list of DynamicStatusIDs, ...) =>	
	<= ACK(SubscriptionHandle)
:	
whenever the value of a subscribed dynamic status parameter changes:	
	<= NotifyEvent(SubscriptionHandle, DynamicStatusID, *value*)
ACK(NULL) =>	
:	
	<= NotifyEvent(SubscriptionHandle, DynamicStatusID, *value*)
ACK(NULL) =>	
:	
when the client no longer wishes to be notified of changes:	
UnsubscribeEvent(SubscriptionHandle) =>	
	<= ACK(NULL)

It may take a very long time before the client is notified of an event after it subscribes to the event. If the client intends to terminate the service session after it sends a SubscribeEvent command, the "life" parameter in the SubscribeEvent command must be set to "persistent". While such a long-term subscription is effective, the availability of the client and the Functional Unit must be periodically checked using the Salutation Manager's Availability Check function so that:

- the client can be assured that the Functional Unit has not forgotten (e.g. by crush or re-start) that the client is still waiting for the notification of the event, and
- the Functional Unit can be assured that the client is still up and running (e.g. not crushed) to receive a notification of the event.

If the "life" parameter in a SubscribeEvent command is set to "session", the subscription is automatically canceled when the service session is terminated. In this case, the "checkInterval" parameter of the SubscribeEvent command must be omitted, and Availability Check is not performed.

Please refer to the relevant sections in the part-1 of the specification for the detail of Salutation Manager Protocol and the Salutation API definitions for the Availability Check. This section describes the flow of operations for the client and the Functional Unit to initiate and end the Availability Check.

When a client application intends to send a SubscribeEvent command with “life = persistent” parameter, it must have registered itself as a [Client] Functional Unit with the Salutation Manager (SLM). In the following description, the [Client] Functional Unit sending a SubscribeEvent is called "Client-FU", and the Functional Unit to which the SubscribeEvent command is sent is called "Server-FU".

1) Before the Client-FU sends the SubscribeEvent to the Server-FU, it calls **slmStartAvailabilityCheck()** with the following parameters:

   - AvailabilityCheckMode = **Receiver**
   - SLM-ID of the SLM with which the Server-FU is registered
   - FU Handle of the Server-FU
   - Check Interval
   - FU Handle of the Client-FU (i.e. its own FU Handle)
   - The entry point of the Client-FU's call-back function which will be called by the SLM when the Server-FU is found to be unavailable.

   slmStartAvailabilityCheck() returns a AvailabilityCheckHandle to the Client-FU. The Client-FU must retain the AvailabilityCheckHandle associated with this SubscribeEvent request, so that when it cancels the SubscribeEvent request later, it can also cancel this Availability Check request.

2) Then, the Client-FU sends the SubscribeEvent to the Server-FU including the following parameters:

   - Check Interval (the same value as that is specified in **slmStartAvailabilityCheck()**)

3) The Server-FU processes the received SubscribeEvent, and returns a response back to the Client-FU.

   If the response is negative (i.e. NACK), the Client-FU must call **slmCancelAvailabilityCheck()** with the AvailabilityCheckHandle which was given by the previous **slmStartAvailabilityCheck()**. Otherwise, proceed to the following.

   If the response is positive (i.e. ACK), the Server-FU calls **slmStartAvailabilityCheck()** with the following parameters:

   - AvailabilityCheckMode = **Sender**
   - SLM-ID of the SLM with which the Client-FU is registered
   - FU Handle of the Client-FU
   - Check Interval : copied from the SubscribeEvent
   - FU Handle of the Server-FU (i.e. its own FU Handle)
   - The entry point of the Server-FU's call-back function which will be called by the SLM when the Client-FU is found to be unavailable.

slmStartAvailabilityCheck() returns a AvailabilityCheckHandle to the Server-FU. The Server-FU must retain the AvailabilityCheckHandle associated with this SubscribeEvent request, so that when the SubscribeEvent request is canceled, it can also cancel this Availability Check request.

4) The Availability Check is performed between the client-side SLM and the server-side SLM (which can be the same SLM). As long as the Client-FU and the Server-FU are available, the call-back function of the Client-FU or the Server-FU is not called.

   If the Server-FU is found to be unavailable, the call-back function of the Client-FU is called. The Client-FU must call **slmCancelAvailabilityCheck()** with the SubscriptionHandle given by the previous **slmStartAvailabilityCheck()**.

   If the Client-FU is found to be unavailable, the call-back function of the Server-FU is called. The Server-FU must call **slmCancelAvailabilityCheck()** with the SubscriptionHandle given by the previous **slmStartAvailabilityCheck()**. The Server-FU invalidates the event subscription from the Client-FU.

5) When the Client-FU no longer needs to be notified of the event, it sends an UnsubscribeEvent command to the Server-FU, and calls **slmCancelAvailabilityCheck()** with the AvailabilityCheckHandle which was given by the previous **slmStartAvailabilityCheck()**.

6) When the Server-FU receives the UnsubscribeEvent command from the Client-FU, it calls **slmCancelAvailabilityCheck()** with the AvailabilityCheckHandle which was given by the previous **slmStartAvailabilityCheck()**.

1.4.5.2.Message Description

1.4.5.2.1.QueryDynamicStatus

The client sends a *QueryDynamicStatus* message to get the current value of a dynamic status parameter. The message contains a Dynamic Status ID as a parameter.

If the specified dynamic status parameter is unknown or not supported by the FU, the command is rejected by NACK.

Response

One of the following is returned in response to this message:

- *ACK*(ANY)

  The server has successfully processed the command and is returning the value of the specified dynamic status parameter.

- *NACK*(ReturnCode)

  The server has failed to process the command. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
QueryDynamicStatus              ::= [APPLICATION tagQueryDynamicStatus] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    dynamicStatusId                 [0] DynamicStatusID
}
```

1.4.5.2.2.SubscribeEvent

The client sends a *SubscribeEvent* message to request the Functional Unit to notify the client of any changes of the value of particular dynamic status parameter(s).

If any of the specified dynamic status parameters are unknown or not supported by the FU, the entire command is rejected by NACK.

"Life" parameter indicates whether the subscription is valid only during the current service session (life = "session") or the subscription should be retained even after the current service session is terminated (life = "persistent"). “job” shall not be set to the life parameter.

If the life of “persistent” is specified in the command, the client application may close the service session after it receives the ACK for the command. The client application that is to receive the event notification must have registered itself with the SLM as a [Client] Functional Unit.

If “life = persistent”, the “checkInterval” parameter must be present. If “life = session”, the “checkInterval” parameter must be omitted.

The event notification is sent to the client application that submitted the *SubscribeEvent* command. Because the Functional Unit Handle of the [Client] FU and the SLM-ID of the SLM with which the [Client] FU is registered are passed to the server FU at the *Open Service* request, the server FU will be able to send an *Open Service* request to the [Client] FU to send the event notification if no service session exists.

If "life = persistent” parameter is specified in the *SubscribeEvent* command, the Functional Unit has to request the Salutation Manager (SLM), by calling slmStartAvailabilityCheck(), to periodically check if the [Client] FU to receive the event notification is still available.

Response

One of the following is returned in response to this message:

- *ACK*(SubscriptionHandle)

  The server has successfully processed the command and is returning a handle that uniquely identifies this particular subscription.

- *NACK*(ReturnCode)

  The server has failed to process the command. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
SubscribeEvent              ::= [APPLICATION tagSubscribeEvent] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    eventList                   [0] SET OF DynamicStatusID,
    life                        [1] Life,
    checkInterval               [2] INTEGER              OPTIONAL
                                    -- Interval (in seconds) for the FU-side SLM to periodically
                                    -- check the availability of the [Client] FU to receive the job
                                    -- status notification
                                    -- This parameter shall be present if life = persistent.
                                    -- This parameter shall be omitted if life = session.
}
```

1.4.5.2.3.NotifyEvent

The Functional Unit sends a *NotifyEvent* message to the client to notify the new value of a dynamic status parameter that the client has previously subscribed to, when the value has changed.

If the indicated SubscriptionHandle or DynamicStatusID is unknown to the client, the message is rejected by NACK.

Response

One of the following is returned in response to this message:

- *ACK*(NULL)

  The client has successfully accepted the notification.

- *NACK*(ReturnCode)

  The client has failed to process the message. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
NotifyEvent                 ::= [APPLICATION tagNotifyEvent] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    subscriptionHandle          [0] SubscriptionHandle,
    dynamicStatusId             [1] DynamicStatusID,
    dynamicStatusValue          [2] ANY
}
```

1.4.5.2.4.UnsubscribeEvent

The client sends an *UnsubscribeEvent* message to the Functional Unit when it no longer wishes to be notified of any changes of the value of subscribed dynamic status parameter(s).

If the specified subscription handle is unknown to the FU, the command is rejected by NACK.

Response

One of the following is returned in response to this message:

- *ACK*(NULL)

The server has successfully processed the command and canceled the specified subscription.

- *NACK*(ReturnCode)

  The server has failed to process the command. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
UnsubscribeEvent            ::= [APPLICATION tagUnsubscribeEvent] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   subscriptionHandle           [0] SubscriptionHandle
}
```

1.4.6.Vendor Escape

1.4.6.1.Overview

There may be cases that vendor-unique commands and responses are necessary to implement vendor-unique functions in each Functional Unit. The VendorEscape message may be used to implement such functions. The Manufacturer, Product, and Version attributes which may be included in any Functional Unit Description Records help vendor-unique Salutation applications to determine if and what vendor-unique functions are supported.

1.4.6.2.Message Description

1.4.6.2.1.VendorEscape

The number and data type of parameters in the VendorEscape message and its response are to be defined by each vendor's specification, and out of the scope of this document.

VendorEscape is unique to vendors, so, the sender of VendorEscape command should check the validity of this command prior to issuing it. The sender can know it by examining the Manufacturer, Product and/or Version in the query capability.

Response

One of the following is returned in response to this message:

- *ACK*(...)

  The server has successfully processed the command.

- *NACK*(ReturnCode)

  The server has failed to process the command. *NACK* includes a **ReturnCode** which indicates the reason of the failure.

ASN.1 Syntax Definition

```
VendorEscape                ::= [APPLICATION tagVendorEscape] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   parameter                    [0] ANY
}
```

1.5.Common Attributes

The following table shows the attributes that are **mandatory** in all the **registered** Functional Unit Description Records regardless of the type of Functional Unit.

Capability Attribute Name	ID	Data Type	Compare Function ID
Major version	10	INTEGER	intEqualTo
Minor version	11	INTEGER	intGreaterThanOrEqualTo
Default coded character set	20	CharSetID	intEqualTo
FU name	30	DisplayString (SIZE(0..63))	strEqualTo
Manufacturer name	40	DisplayString (SIZE(0..63))	strEqualTo
Manufacturer product name	41	DisplayString (SIZE(0..63))	strEqualTo
Manufacturer product version	42	DisplayString (SIZE(0..63))	strEqualTo
Physical location	50	DisplayString (SIZE(0..255))	strEqualTo
Contact person name	51	DisplayString (SIZE(0..255))	strEqualTo
Authentication flavors	60	SET OF AuthenticationFlavor	setIntDoesContain

Note : Major/minor version attributes indicate the version number of the Salutation Personality Protocol architecture definition of the Functional Unit. When these two attributes are included in a requested FUDR, the requested FUDR matches the registered FUDR if the two FUDRs have the same major version number AND the minor version number of the registered FUDR is greater than or equal to that of the requested FUDR.

Under the initial version of the Salutation Architecture, the following values must be specified in any registered FUDRs:

- Major version = 2
- Minor version = 0

Additional attributes are defined for each type of Functional Unit. All the defined attributes must be present in a registered FUDR.

1.6.National Language Support

This section defines how character sets are encoded under the Salutation Architecture. Any other internationalization considerations are discussed under the specification of each Functional Unit as required.

The following textual information are subject to the definition in this section.

- **Attribute** values of textual data type in the Functional Unit Description Records
- **Parameter**s of textual data type in commands and responses under the Salutation Personality Protocol.
- **“Data”** of textual data type, e.g. plain text, unless the coded character set is identified by another method, e.g.
    - □ the architecture/specification of data format defines its own coded character set

identification rule, or

□ coded character set identification rule is defined by the specification of Functional Unit

For example, versit's **vCard** data format has its own coded character set identifier, therefore, this is outside of the scope of this section's definition. Refer to the Versit documents listed in the "References" section on page 275 for the detail of versit's vCard data format.

These are called target attributes, parameters and "data" in the following description.

Note: The architecture only defines the coded character set identification rule for interchange. Implementations may choose whichever coded character set they like for internal usage.

DisplayString data type is defined by the architecture as follows. It should be used for any textual data definition for the target attributes and parameters.

```
DisplayString                    ::= OCTET STRING          -- Textual information
```

CharSetID data type is defined by the architecture as follows to identify a particular coded character set.

```
CharSetID                        ::= INTEGER       -- Coded character set ID as registered with IANA
                                                   -- (MIB enumeration value is used)
```

The value of CharSetID is an integer registered with the Internet Assigned Numbers Authority (IANA) as MIB enumeration value to identify a coded character set. The IANA's registry of the coded character sets can be found at:

ftp://venera.isi.edu/in-notes/iana/assignments/character-sets

The architecture recommends implementations to choose coded character sets for interchange from the following list:

Coded Character Set	Countries	IANA-registered character set name	Value of CharSetID
ISO 8859-1	Latin-1 countries	ISO_8859-1:1987	4
ISO 10646-1 level 3 (Unicode)	Countries using double-byte characters	ISO-10646-UCS-2	1000
JIS X0208-1978 (Shift JIS)	Japan	Shift_JIS	17

Note: This list may grow based on requirements.

The following attribute is defined for the Functional Unit Description Record (FUDR):

- Default Coded Character Set

Data Type : CharSetID

All the target attributes of DisplayString data type in the FUDR are encoded in the coded character set specified by this attribute.

1.7.[Client] Functional Unit

1.7.1.Overview

In general, Functional Units are abstraction of server applications and provide services to client applications or other Functional Units. However, a special Functional Unit, called **[Client] Functional Unit**, is defined to represent certain client applications rather than server applications.

The [Client] Functional Unit is semantically different from other Functional Units as it represents a client, not server, application. However, the Salutation Manager treats it just like any other Functional Units unless otherwise stated. For example, [Client] Functional Units will be included in a *Query Capability* reply message with all the other Functional Units if a *Query Capability* call message with the Wild Functional Unit ID is received.

A client application must register its capability as a [Client] Functional Unit to the Salutation Manager only if it expects to receive an *Open Service* request from another Functional Unit. A client application may receive an *Open Service* request from another Functional Unit under the following cases:

- The client application sends a job-request-type command to a Functional Unit, which requests any of the following, and then terminates the session.
  - □ The life of the job is "job" or "persistent", and the command requested to notify the client of certain job (entry) status change(s).
  - □ The command requested a delayed-mode data transfer to and/or from the client.
- The client application sends a SubscribeEvent command to a Functional Unit with "life = persistent" parameter, and then terminates the session.

In these cases, the Functional Unit sends an *Open Service* to the client before sending a NotifyJob(Entry)Status, RequestDataTransfer, or NotifyEvent message. It sends a *Close Service* after the message sequence is completed.

The “Personality Protocol ID” parameter of *Open Service* request to the [Client] FU shall be one (1).

1.7.2.Attributes

The following capability attribute is defined for the [Client] Functional Unit.

- User ID

  It contains the User ID of the user associated with the client application. The network login name is recommended to be used as the User ID if applicable, however implementation may choose to use another scheme, for example, the full name of the user, or even the user class name such as “general” or ”administrator”.

  It shall be NULL if no user is associated with the client application.

Refer to the “User Identification and Authentication” section in the part-1 of the architecture specification for the detail of this attribute.

When a client application registers the [Client] Functional Unit Description Record to the Salutation Manager, the following common capability attributes are also included to describe the client application.

- Major version
- Minor version
- Default coded character set
- FU name
- Manufacturer name
- Manufacturer product name
- Manufacturer product version
- Physical location
- Contact person name
- Authentication flavors

No command attribute for the attribute repository is defined for the [Client] Functional Unit.

1.7.3.Dynamic Status Parameters

No dynamic status parameter is defined for the [Client] Functional Unit.

1.7.4.Messages

The [Client] Functional Unit may receive the following messages.

- NotifyJobStatus
- NotifyJobEntryStatus
- NotifyEvent
- RequestDataTransfer
- RequestNextData

The [Client] Functional Unit may send the following messages:

- ACK
- NACK
- DataBlockDescription
- TransferDataBlock

2.Document Systems

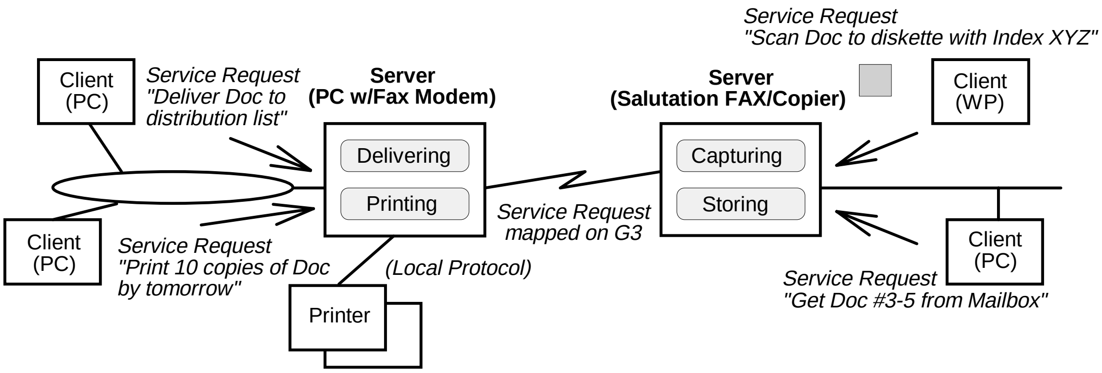
2.1.Document Systems Overview

2.1.1.Architecture Scope

The abstract document system defined in Salutation Architecture (hereafter termed as "Salutation document systems") provides "on-line" document operational capabilities with the following service functions.

- capturing document
- manipulating document
- storing document
- delivering document
- printing document

The following figure illustrates a typical configuration of Salutation document systems and outlines the architecture scope.


Service Request
"Scan Doc to diskette with Index XYZ"
Client (PC)
Service Request "Deliver Doc to distribution list"
Server (PC w/Fax Modem)
Server (Salutation FAX/Copier)
Client (WP)
Delivering
Printing
Capturing
Storing
Service Request mapped on G3
Client (PC)
Service Request "Print 10 copies of Doc by tomorrow"
(Local Protocol)
Client (PC)
Service Request "Get Doc #3-5 from Mailbox"
Printer

The following bullets summarize the key requirements and the consideration points in defining the architecture for Salutation document systems.

- Server (service provider) is considered as a common resource with some sort of intelligence, rather than a host controlled peripheral.
- Supports both batch type service request and interactive type service request. Service request can either be queued or be executed on-the-fly.
- Service requests may be issued from multiple clients at the same time.
- Service requests are defined independent of transports. Recommended mapping on to G3 facsimile protocol needs to be described.
- Functional split between clients and servers will be flexibly configured, to some extent, in accordance with the implementation level of each service.

The architecture focuses on defining the Functional Units which provide the following services within Salutation document systems.

- Document printing services : termed as [Print] Functional Unit
- Document delivering services via fax protocol : termed as [Fax Data Send] Functional Unit
- Document and data storing services : termed as [DOC Storage] Functional Unit

2.1.2.Common Characteristics in Document Systems

Abstract Functional Units in Salutation document systems will share many of the characteristics in common. This section describes the following common characteristics.

- Document content descriptors (or attributes), and the default interchange format
- Document transfer procedure between client and server applications

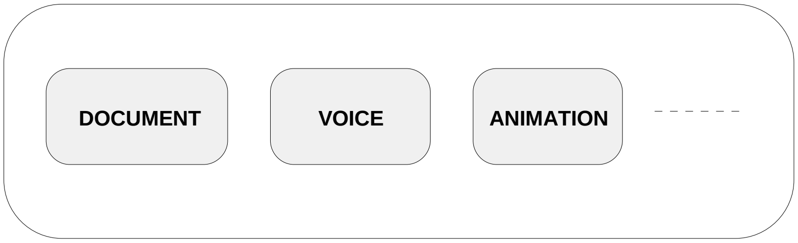
2.1.2.1.Document Content Descriptor

2.1.2.1.1.Structure and Attributes

The term "Content Data" represents a meaningful unit of data sequence, such as compound document, file, etc., which is treated in a certain consistent manner by client/server applications with Salutation Personality Protocol.

Content data is constructed of a single or multiple data elements of diverse characteristics. Here, each "data element" is considered to be defined as a simple subset of content data type, in such a way as "(printable/visible) document" content data, "voice" content data, "animation" content data, etc.

CONTENT DATA



In order to describe content data which is constructed of multiple data elements of diverse nature, it is required to define another descriptor which specifies construction type (such as SGML, MIME, ScriptX, etc., or more simply SET{doc, voice, animation}). This release of the architecture does not cover those constructed content data types.

2.1.2.1.2.Mode of Content Data Operation

The following modes of operation are defined for content data operation. Note that it does not describe the characteristics of content data itself, but does describe how it should be handled by client/server applications.

- Transparent Mode

  Content data is handled as a bulk data without interpreting its content

- Non-Transparent Mode

  Content data is handled by interpreting its content

Most of content data handled by Salutation document systems are document data. However, when they are operated in transparent mode, content data other than document data can be handled in Salutation document systems.

Operation modes of content data are implicitly determined by the value of the correspondent attributes, or by the nature of the Functional Unit.

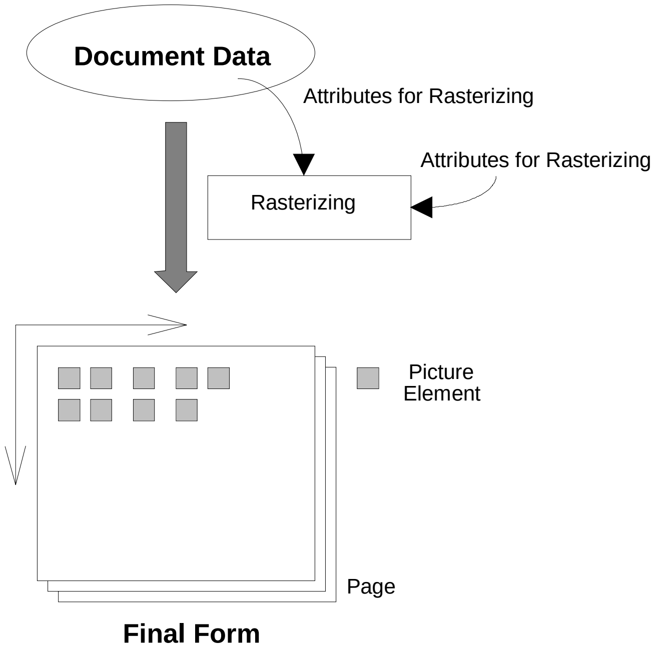
Functional Unit	Transparent Mode	Non-Transparent Mode
[Print]	(Not Applicable)	Default
[Fax Data Send]	faxProtocol = BFT Protocol	faxProtocol = G3 Protocol
[DOC Storage]	modeOfStore = File Data Mode	modeOfStore = Doc Data Mode

Transparent mode is implied when File data mode is selected for [DOC Storage] Functional Unit.

2.1.2.1.3.Definition of Document Data

Document data is a meaningful unit of data sequence which can be transformed (rasterized) into a set of picture element data mapped onto two-dimensional presentation space. More simply, it is digital data representation of printable, or visible sequence of pages.

The term "rasterizing" is defined as a process to transform document data into a set of picture element data mapped onto two dimensional presentation space (such as memory space) by referring either internal attributes of document data or external attributes set as a parameter of service request commands.



2.1.2.1.3.1.Document Data Format

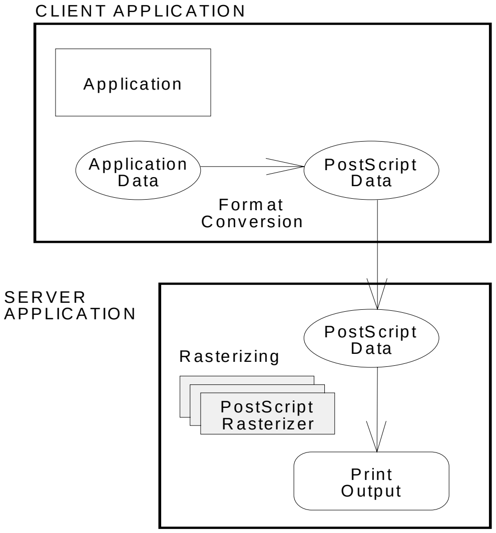
Document data is categorized as follows in accordance with each format type. The grouping is just for clarification, and need not be incorporated in the architecture as an attribute to describe document data types.

Type Description	Format	External Attributes Setting
Image Bitstream	Bi-Level	Yes (fill order, resolution, etc., ..)
	Grayscale	Yes
	Color	Yes
Structured Image Data	MS-53A12 TIFF BMP PCX DCX :	No (required attributes will be contained within document data)
Printer Language (including PDL)	PS ESC/P :	No (required attributes will be contained within document data)
Text	Plain-Text :	Yes (character set, etc., ..)
Others	(for further study)	(for further study)

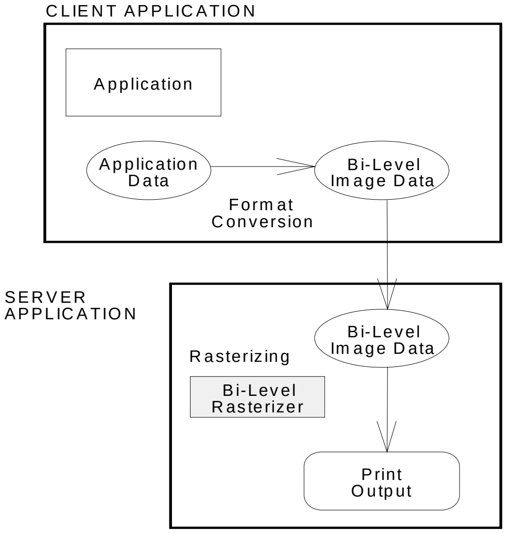
Default Interchange Format and Compatibility Considerations

Types of document data which can be handled within Salutation document systems depend upon the capabilities of each implemented Functional Unit. In order to assure the minimum level of compatibility (document data interchange-ability) across Salutation document systems, it is recommended that all the compliant systems, when in a non-transparent operation mode, support the common data format, e.g. bi-level image bit-stream, as described in the following sections. The common data format will be dined in the next release.

A client application should negotiate with a server application on what kind of document data it can handle and prepare appropriate document data from original data by applying "format conversion" when required.



Server with PostScript Rasterizing Capability



Server with Minimum Rasterizing Capability

2.1.2.1.3.2.Document Data Attributes

Document data format attributes indicate the formats of document data in several contexts of Salutation document systems.

The following enumerated values are assigned to document data formats.

```
DataFormat                    ::= ENUMERATED
{
    notSpecified                  (127),
--Document Related Data Format Start
--
-- bi-Level Image Bitstream listed bellow
-- When data format is "biLevelImageStream", "biLevelImageStreamAxisSize",
-- "biLevelImageStreamTotalSize", or "biLevelImageStreamPageDimension", "ImageStreamAttributes",
-- must be referred or specified.

    biLevelImageStream              (1000),         --Three ImageSize types are supported
                                                    -- When this data format is set, the image size
                                                    -- attribute can be selected from "axisSize",
                                                    -- "totalSize" or "pageDimensions".
    biLevelImageStreamAxisSize      (1001),         -- axisSize in ImageSize is supported.
                                                    -- When this data format is set, the image size
                                                    -- attribute must be "axisSize".
    biLevelImageStreamTotalSize     (1002),         -- totalSize in ImageSize is supported.
                                                    -- When this data format is set, the image size
                                                    -- attribute must be "totalSize".
    biLevelImageStreamPageDimension (1003),         -- pageDimension in imageSize is supported.
                                                    -- When this data format is set, the image size
                                                    -- attribute must be "pageDimensions".

-- Structured Image Data listed bellow
    ms53A12                         (1010),
    tiff                            (1011),
    bmp                             (1012),
    pcx                             (1013),
    dcx                             (1014),
    winMetaFile                     (1015),
    os2MetaFile                     (1016),
    xdw                             (1017),         -- DocuWorks image format. Fuji Xerox Co. Ltd.
    jfif                            (1018),         -- Color image format
```

```
-- Printer Datastream listed bellow.
-- Each PDL needs the version information. PDL   version will be further studied.
    langPCL                    (1203),      -- Printer Control Language. Hewlett-Packard Co.
    lang201PL                  (1204),      -- NEC Co.
    langPJL                    (1205),      -- Printer Job Language. Hewlett-Packard Co.
    langPS                     (1206),      -- PostScript(TM). Post Script is a trademark of
                                            -- Adobe Systems Inc.
    langEscapeP                (1209),      -- EPSON ESC/P(TM). Epson Co.
    langLIPS                   (1239),      -- LBP Image Processing System. Canon Inc.
    langIPDS                   (1250),      -- Intelligent Printer Data Stream,
                                            -- IBM Printing Systems. Corresponds to
                                            -- langIPDS(7) of RFC1759.
    langPAGES                  (1251),      -- Page Printer Advanced Graphics Escape Set.
                                            -- IBM Japan Ltd.
    langMODCA                  (1252),      -- Mixed Object Document Content Architecture,
                                            -- IBM Printing Systems. Corresponds to
                                            -- langIPDS(15) of RFC1759.
    langRPDL                   (1260),      -- Ricoh   Corp.
    langART                    (1270),      -- Fuji Xerox Co. Ltd.

-- Unstructured Text Data listed bellow. (for further study)
    plainText                  (1400),
--
-- Structured Text Data (for further study)
--
-- Portable Document
    pdf                        (1600)       -- Portable Document Format,
                                            -- Adobe Systems Inc.
-- Other Types (for further study)
--
-- Document Related Data Format End
}


ImageStreamAttributes         ::= SEQUENCE
{
                                   -- All parameters shall be specified for “biLevelImageStream”
                                   -- document format.
    imageSize                  [0] ImageSize,
    imageCompAlgorithm         [1] ImageCompAlgorithm,
    imageByteFillOrder         [2] ByteFillOrder,
    imageResolution            [3] ImageResolution
}
```

```
ImageSize                       ::= CHOICE
{
    axisSize                        [0] SEQUENCE
    {
        xAxisSize                   [0] INTEGER,    -- Unit : dot
        yAxisSize                   [1] INTEGER     -- Unit : dot
    },
    totalSize                       [1] INTEGER,    -- Unit : byte
    pageDimensions                  [2] PageDimensions
}

PageDimensions                  ::= SEQUENCE
{
    recordingWidth                  [0] RecordingWidth,
    maximumRecordingLength          [1] MaximumRecordingLength
}

RecordingWidth                  ::= ENUMERATED
{
    rw864                           (0),
    rw1216                          (1),
    rw1728                          (2),
    rw2048                          (3),
    rw2432                          (4)
}

MaximumRecordingLength          ::= ENUMERATED
{
    a4                              (0),
    b4                              (1),
    unlimited                       (2)
}

ImageCompAlgorithm              ::= ENUMERATED
{
-- Following value is meaningful when document data format is biLevelImageStream or tiff.
    raw                             (0),
    mh                              (1),
    mhb                             (2),            -- EOL Byte Boundary
    mr                              (3),
    mrb                             (4),            -- EOL Byte Boundary
    mmr                             (5),
    jpeg                            (6),            -- Compression for color image
    jbig                            (7),            -- Progressive Bi-level Image Compression
                                                    -- ITU-T Recommendation T.82
    other                           (127)
}
```

```
ByteFillOrder                  ::= ENUMERATED
{
-- Following value is meaningful when document data format is biLevelImageStream or tiff.
-- ByteFillOrder shows the bit order in the image data byte.
-- When image data is raw data (not compressed), it shows the Byte Fill Order of raw image
-- data. When image data is compressed, it shows the Byte Fill Order of compressed data.

--      addr0   addr1   addr2   addr3
--      [0..7]  [8..15] [16..23][24..31] ..  ByteFillOrder=msb case
--      [7..0]  [15..8] [23..16][31..24] ..  ByteFillOrder=lsb case

   msb                           (0),
   lsb                           (1)
}

ImageResolution                ::= ENUMERATED
{
-- Following value is meaningful when document data format is biLevelImageStream , tiff, bmp, pcx
-- or dcx.
   normal                        (0),            -- 8x3.85l/mm
   fine                          (1),            -- 8x7.7l/mm
   semi-superFine                (2),            -- 8x15.4l/mm
   superFine                     (3),            -- 16x15.4l/mm
   dpi180                        (4),            -- 180dpi
   dpi200                        (5),            -- 200dpi
   dpi240                        (6),            -- 240dpi
   dpi300                        (7),            -- 300dpi
   dpi360                        (8),            -- 360dpi
   dpi400                        (9),            -- 400dpi
   dpi600                        (10),           -- 600dpi
   dpi720                        (11),           -- 720dpi
   dpi800                        (12),           -- 800dpi
   dpi1200                       (13),           -- 1200dpi
   dpi200x100                    (30),           -- 200x100dpi G4 Optional
   dpi100                        (31)            -- 100dpi
}
```

2.1.2.2.Document Transfer Procedure

This section describes document transfer procedure defined for Salutation document systems. Document transfer procedure follows the definition of Data Transfer Message Sequence described in "**Data Transfer** Messages" section on page 18.

Some Functional Units will provide capability of spooling series of job requests and content data, others will not. The document transfer procedure covers both Functional Units with a spooler and without a spooler.

2.1.2.2.1.Typical Scenarios and Flow Diagrams

To start with, typical example scenarios are given :

- Printing document to **[Print]** Functional Unit in copier (Client to Server)

  A user completes a sales report using a word processor running on a PC. The user needs ten hard copies of the report for circulation to related departments. The user drags and drops the report to a Salutation copier icon on a PC screen. A client application on the PC sends "Print" command request via LAN network to [Print] Functional Unit embedded in a Salutation copier on another floor.

- Sending document to **[Fax Data Send]** Functional Unit in fax server (Client to Server)

  He now is ready to send the report to his business partners after his updating the reports on the PC. He again drags and drops the updated report to a fax icon on a screen with specifying the list of partners' names and phone numbers. A client application on the PC sends "Send Fax" command request via network to [Fax Data Send] Functional Unit in a fax machine. The report will be faxed from him to those who locate in the partners' offices.

- Retrieving document from **[DOC Storage]** Functional Unit in fax (Server to Client)

  [DOC Storage] Functional Unit in the fax spools received documents for client access. A notification message of document receipt will appear on the client PC. He browses the received documents on PC screen and then selects a document in the list by double-clicking the document. A client application on the PC sends a request to [DOC Storage] Functional Unit in the fax to retrieve a document into the PC local storage.

2.1.2.2.2.Guidelines for Applying Data Transfer Message Sequence

The following bullets summarize the guidelines for applying Data Transfer Message Sequence defined in "**Data Transfer** Messages" on page 18.

- In case that the format of document data is bi-level image stream and when in a non-transparent operation mode, the document data shall be divided into multiple data blocks so that each data block contains one "page" of the document. The begin/end data block flags of TransferDataBlock command described on page 29 are used to indicate the page boundaries.

Client	**Server**

Document Transfer Message Sequence

DataBlockDescription(BiLevelImageStream, ..)=>

<=RequestNextData

Sending First Page of Document

TransferDataBlock(Begin, End)=>

<=RequestNextData

Sending 2nd Page of Document

TransferDataBlock(Begin, End)=>

<=RequestNextData

Sending 3rd (last) Page of Document

TransferDataBlock(Begin, End, Last)=>

<=ACK

- Each data block may be segmented freely by application.

Client	**Server**

:

Sending First Segment of Data Block

TransferDataBlock(Begin)=>

<=RequestNextData

Sending 2nd Segment of Data Block

TransferDataBlock()=>

<=RequestNextData

Sending 3rd (last) Segment of Data Block

TransferDataBlock(End)=>

<=RequestNextData

:

However, for the sake of simplicity, examples shown below assume a whole document data is transferred in one TransferDataBlock command.

- **Use of DataBlockDescription Message**

DataBlockDescription is used to convey information associated with document data, such as document data format descriptor and other attributes required for rasterizing / interpretation. Within Salutation document systems, DataBlockDescription message takes the constructed parameter "documentDataDescriptor" as defined bellow.

```
DocumentDataDescriptor              ::= SEQUENCE
{
    documentDataFormat                  [0] DataFormat,
    documentFormatInterpretation        [1] DocFormatInterpretation         OPTIONAL
}

DocFormatInterpretation             ::= CHOICE
{
    imageStreamAttributes               [0] ImageStreamAttributes
                                            -- Chosen when documentDataFormat is
                                            -- biLevelImageStream.
                                            --
                                            -- Other interpretation attributes are for
                                            -- further study.
}
```

When a document is transferred, a *DataBlockDescription* message must be included before the first *TransferDataBlock* message in a Data Transfer Message Sequence so that the receiver will be able to know the format of the document. Additional *DataBlockDescription* messages must be inserted as required if the document consists of data blocks in multiple formats.

- When a document of bi-level image stream format is transferred, all parameters of “ImageStreamAttributes” must be specified within the “DocumentDataDescriptor” parameter of the *DataBlockDescription* message so that the receiver will be able to interpret the document data correctly. If the document data is transferred from a URL-specified data location, the “DocumentDataDescriptor” parameter must be included in the command that initiates the document data transfer because *DataBlockDescription* message cannot be used since the document data are transferred under a protocol out side of the Salutation architecture.

2.1.2.2.3.Requesting Document Transfer between Functional Units

A client can also request document transfer between Functional Units.

Typical scenario is that a client requests [Print] Functional Unit to print a document stored in [DOC Storage] Functional Unit. The following procedure is applied.

Client	**Server**

Establish Service Session to Access [DOC Storage] Functional Unit

OpenService(DOC Storage) =>

Job Request Message Sequence **-** *Retrieve Document from DOC Storage*

RetrieveDoc(DocumentID, DataDestination=ExportPool) =>

<= ACK(DataHandle)

Close Session with [DOC Storage] Functional Unit

CloseService =>

--

Establish Service Session to Access [Print] Functional Unit

OpenService(Print) =>

Job Request Message Sequence **-** *Print Document*

Print(.., DataSource=AbsoluteFunctionalUnitHandle, DataHandle) =>

<= ACK(..)

Close Session with [Print] Functional Unit

CloseService =>

--

[Print] Establishes Service Session with [DOC Storage]

OpenService(DOC Storage) =>

Job Request Message Sequence **-** *Print Document*

RequestDataTransfer(DataHandle) =>

<= DataBlockDescription

RequestNextData =>

<= TransferDataBlock

ACK(NULL) =>

[Print] Closes Session with [DOC Storage]

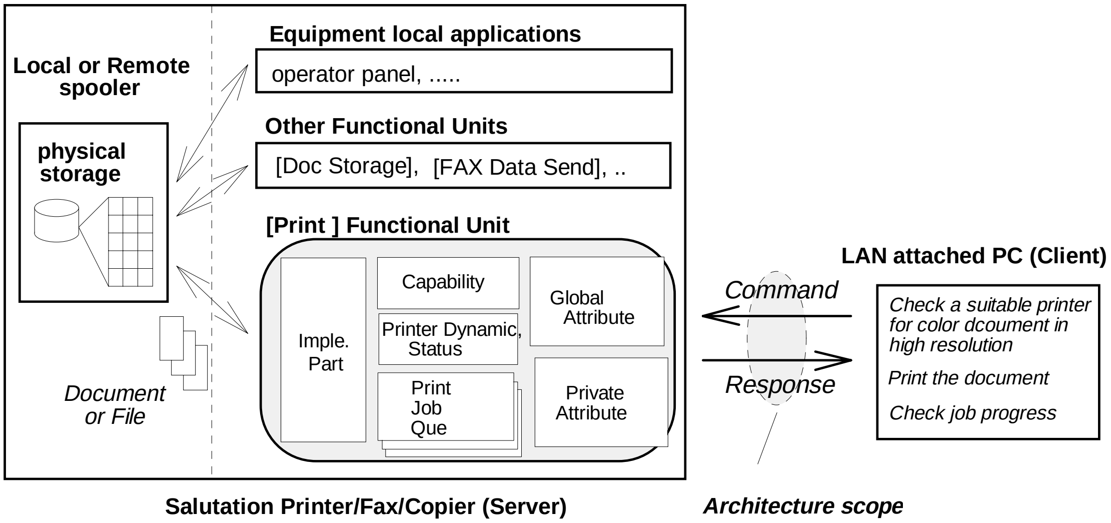
CloseService =>

2.2.[Print] Functional Unit

2.2.1.Overview

[Print] Functional Unit provides a printing service for a client user to print documents on local or remote printing equipment and also to query various status of printing jobs and equipment. The client user can select suitable equipment for the print job by inquiring the equipment capability provided by the attributes of **[Print]** Functional Unit. **[Print]** Functional Unit can be found typically on LAN attached printer or copier or fax equipment.

The following figure illustrates a configuration model to understand how **[Print]** Functional Unit works with other resources within equipment and with remote client who issues service request commands.



[Print] Functional Unit is considered to be composed of the following logical sub-components or service elements.

- Capability Attribute
- Global Attribute
- Private Attribute
- Printer Dynamic Status
- Print Job Queue

2.2.2.List of Functional Unit Attributes

The following table describes the attributes defined for [Print] Functional Unit, and specifies what protocol data unit will use those attributes. A ChangeJobAttribute can apply to the Attributes, which have a "Data Type as Command Attribute" and are included in Print command.

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute (default[1])	Private/ Job Attribute
personalityProtocol	10000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No/No
supportedCommand	10001	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No/No
dynamicStatusId	10002	N/A	SET OF DynamicStatusID (setIntDoesContain)	No	No/No
spoolStorage	10003	N/A	SpoolStorage (boolEqualTo)	No	No/No
minimumCheckInterval -- the minimum allowed -- value to be set in the -- checkInterval parameter -- of a SubscribeEvent -- command	10004	N/A	INTEGER (intGreaterThanOrEqualTo)	No	No/No
documentFormat	10010	DataFormat	SET OF DataFormat (setIntDoesContain)	No	No/No
imageCompAlgorithm	10011	ImageCompAlgorithm	SET OF ImageCompAlgorithm (setIntDoesContain)	No	No/No
imageByteFillOrder	10012	ByteFillOrder	SET OF ByteFillOrder (setIntDoesContain)	No	No/No
imageResolution	10013	ImageResolution	SET OF ImageResolution (setIntDoesContain)	No	No/No
printPaperSize	10020	PaperSize	SET OF PaperSize (setIntDoesContain)	Yes	No/No
printResolution	10021	ImageResolution	SET OF ImageResolution (setIntDoesContain)	Yes	No/No
printPaperDirection	10022	PaperDirection	SET OF PaperDirection (setIntDoesContain)	Yes	No/No
printCopyCount	10023	INTEGER	INTEGER -- max value (intGreaterThanOrEqualTo)	No (1)	No/No
printPaperInputSelect	10024	PrintPaperInputSelect	SET OF PrintPaperInputSelect (setIntDoesContain)	Yes	No/No
printPaperOutputSelect	10025	PrintPaperOutputSelect	SET OF PrintPaperOutputSelect (setIntDoesContain)	Yes	No/No

[1] Implementation default values to be referred to when neither command parameter nor Private Attribute value is set.

printOutputBinSelect	10026	PrintOutputBinSelect	PrintOutputBinSelect —maximum bin# (intGreaterThanOrEqualTo)	Yes	No/No
printDuplexMode	10027	PrintDuplexMode	SET OF PrintDuplexModeSelect (setIntDoesContain)	Yes	No/No
maximumBindingMargin	10028	INTEGER	INTEGER -- max value (intGreaterThanOrEqualTo)	Yes	No/No
printFaceUpMode	10029	PrintFaceUpMode	SET OF PrintFaceUpMode (setIntDoesContain)	Yes	No/No
printStaplingSelect	10030	PrintStaplingSelect	SET OF PrintStaplingSelec(setIntDoesContain)	Yes	No/No
printPriority	10040	SimpleJobPriority	SET OF SimpleJobPriority (setIntDoesContain)	Yes	No/Yes
modeOfDataTransfer[2]	10041	DataTransferMode	SET OF DataTransferMode (setIntDoesContain)	Yes	No/No
dataLocationScheme	10042	N/A	SET OF DataLocationScheme (setIntDoesContain)	No	No/No
dataTransferTimeOutSettable	10043	N/A	BOOLEAN (boolEqualTo)	No	No/No
dataTransferTimeOutLength -- length in seconds for the FU -- to wait for the next message -- during a data transfer -- message sequence before -- detecting time-out exception	10044	INTEGER (N/A, if the previous dataTransferTimeOutSettable attribute is FALSE) -- Global attribute indicates the -- default length. If the global -- attribute value is zero, the -- default length is not fixed or -- unknown. -- If the private attribute value is -- set to zero, the FU should -- wait as long as possible. -- However, use of zero should -- be avoided.	INTEGER (intGreaterThanOrEqualTo) -- if 0, not fixed or unknown -- (use of 0 should be avoided)	Yes (No, if the previous attribute is FALSE)	Yes/No (No, if the previous attribute is FALSE)

The following enhancements are to be considered for [Print] Functional Unit.

- Version management of [Print] Functional Unit. For evolving command parameters and attributes, mechanism should be defined to identify and to handle multiple versions of specifications.
- Attribute enhancement to identify PDL level and version.

2.2.3.Message & Protocol

This section describes service request protocol for [Print] when the Salutation Personality Protocol is selected.

[2] When “spoolStorage” = FALSE, only “delayed” mode is allowed for this attribute.

2.2.3.1.Document Data Transfer Request

The following commands and responses are used for the Document Data Transfer Request procedure to print a document. Abstract syntax definition of common protocol data unit and its usage for document transfer procedure are described in either "**Data Transfer Messages**" section on page 18 or "

Document Transfer Procedure" section on page 69.

- [Print] FU Mandatory support Command
  - □ Print
- [Print] FU Optional support Command
  - □ VendorEscape
- [Print] FU Mandatory support Common Commands and Responses
  - □ RequestDataTransfer
  - □ DataBlockDescription
  - □ TransferDataBlock
  - □ RequestNextData
  - □ ACK and NACK

2.2.3.1.1.Print Request

Print command is used to request [Print] Functional Unit to print a document data.

ASN.1 Syntax Definition

```
Print                         ::= [APPLICATION tagPrint] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
    modeOfDataTransfer            [0] DataTransferMode          OPTIONAL,
                                      -- Override Global/Private Attribute
    dataSource                    [1] DataLocation              DEFAULT client,
    dataHandle                    [2] DataHandle                OPTIONAL,
                                      -- Omitted in immediate mode data transfer from client, or
                                      -- if the source data location is specified by URL
    inputDocumentFormat           [3] DocumentDataDescriptor    OPTIONAL,
                                      -- Present if and only if dataSource = url
    life                          [4] Life                      DEFAULT job,
                                      -- Specify how long FU should keep a job status:
                                      -- for job life or for session life or persistently.
    jobStatusNotificationMode     [5] JobStatusNotificationMode OPTIONAL,
                                      -- If omitted, no notification is made.
    notificationScheme            [6] NotificationScheme        OPTIONAL,
                                      -- Omitted unless the job status notifications are to be
                                      -- sent to a [Client] FU other than the client that is
                                      -- sending this command
    printControlAttribute         [7] PrintControlAttribute     OPTIONAL
}

PrintControlAttribute         ::= SEQUENCE
{
    printPaperSize                [0] PaperSize                 OPTIONAL,
    printResolution               [1] ImageResolution           OPTIONAL,
    printPaperDirection           [2] PaperDirection            OPTIONAL,
    printCopyCount                [3] INTEGER                   OPTIONAL,
    printPaperInputSelect         [4] PrintPaperInputSelect     OPTIONAL,
    printPaperOutputSelect        [5] PrintPaperOutputSelect    OPTIONAL,
    printOutputBinSelect          [6] PrintOutputBinSelect      OPTIONAL,
    printDuplexMode               [7] PrintDuplexMode           OPTIONAL,
    printFaceUpMode               [8] PrintFaceUpMode           OPTIONAL,
    printPriority                 [9] SimpleJobPriority         OPTIONAL,
    printStaplingSelect           [10] PrintStaplingSelect      OPTIONAL,
    printFileName                 [11] DisplayString            OPTIONAL
}
```

ACK Response

Parameter Name	**Data Type**	**Note**
parameter1	JobHandle	

NACK Response

Name	Description	ReturnCode
rcInvalidModeOfDataTransfer	modeOfDataTransfer is incorrect or not supported	129
rcInvalidDataSource	dataSource is incorrect or not supported	130
rcInvalidDataHandle	dataHandle is incorrect	131
rcInvalidInputDocumentFormat	inputDocumentFormat is incorrect or not supported	132
rcInvalidLife	life is incorrect or not supported	133
rcInvalidJobStatusNotificationMode	jobStatusNotificationMode is incorrect or not supported	134
rcInvalidNotificationScheme	notificationScheme is incorrect or not supported	135
rcInvalidPaperSize	printPaperSize is incorrect or not supported	136
rcInvalidResolution	printResolution is incorrect or not supported	137
rcInvalidPaperDirection	printPaperDirection is incorrect or not supported	138
rcInvalidCopyCount	printCopyCount is incorrect or not supported	139
rcInvalidPaperInputSelect	printPaperInputSelect is incorrect or not supported	140
rcInvalidPaperOutputSelect	printPaperOutputSelect is incorrect or not supported	141
rcInvalidOutputBinSelect	printOutputBinSelect is incorrect or not supported	142
rcInvalidDuplexMode	printDuplexMode is incorrect or not supported	143
rcInvalidFaceUpMode	printFaceUpMode is incorrect or not supported	144
rcInvalidPriority	printPriority is incorrect or not supported	145
rcInvalidStaplingSelect	printStaplingSelect is incorrect or not supported	146

2.2.3.2.Attribute Operations

The following command and response are used for attribute controls. The usage of those commands and responses are described in "**Attribute Repository Messages**" section on page 30.

- [Print] FU Mandatory support common Command
  - □ GetPrivateAttribute
  - □ GetGlobalAttribute
  - □ SetPrivateAttribute
  - □ ACK and NACK

Attributes affected by the above commands are listed in "List of Functional Unit Attributes" section.

2.2.3.3.Dynamic Status Operations

Dynamic Status operations allow a client to know the aspect of Functional Unit and any transition in the aspects. **Dynamic Status Parameter** describes the aspects, for example, noPaper. A client may query the current values of Dynamic Status Parameter, or request [Print] to notify an Event when any transition occurs in the values of Dynamic Status Parameter.

The following commands and response are used for dynamic status operations. The usage of those commands and responses are described in "**Dynamic Status Messages**" section on page 49.

- [Print] FU Mandatory support common Command
  - □ QueryDynamicStatus
  - □ ACK and NACK
- [Print] FU Optional support Command
  - □ SubscribeEvent, UnsubscribeEvent, and NotifyEvent (These commands belong to the same Optional Group, so an FU must support all these commands if it supports them.)

The following Dynamic Status Parameters are defined for [Print] Functional Unit.

Dynamic Status Parameter	**Query**	**Event**	**ID**	**Description**
PrinterOperationStatus	Yes	Yes	10000	status of printing equipment.
PrinterErrorDetail	Yes	No	10001	detail error information of equipment's.
FreeStorageSize	Yes	No	10002	available storage size, K Byte.
PrinterPaperInputTray	Yes	No	10003	status of paper size and direction in each input tray.
ListExcerptPrintJob	Yes	Yes	10004	lists a excerpt from print job descriptions

Data Type of Dynamic Status Parameter

PrinterOperationStatus ::= SET OF PrinterStatusCode

```
PrinterStatusCode                   ::= ENUMERATED
{
    noPaper                             (0),
    noToner                             (1),
    doorOpen                            (2),
    jammed                              (3),
    offline                             (4),
    receiving                           (5),
    error                               (6),
    normal                              (7),
    paperNearEnd                        (8),
    tonerNearEnd                        (9),
    fatalError                          (10),                   -- errors requiring equipment repair
    others                              (127)
}
                                    -- PrinterStatusCode may change from normal to noPaper and
                                    -- doorOpen at the same time. Then the [Print]FU issues
                                    -- NotifyEvent(SubscriptionHandle, DynamicStatusID, SET OF
                                    -- PrinterStatusCode). The PrinterStatusCodes are noPaper and
                                    -- doorOpen. If the doorOpen returns to normal, again the [Print]FU
                                    -- issues NotifyEvent(SubscriptionHandle, DynamicStatusID, SET OF
                                    -- PrinterStatusCode). The PrinterStatusCodes is noPaper only.

PrinterErrorDetail                  ::= SET OF PrinterErrorDescription

PrinterErrorDescription             ::= SEQUENCE
{
    printerStatusCode               :   [0] PrinterStatusCode,
    systemError                         [1] DisplayString,      -- detail description
    others                              [2] DisplayString-- detail description
}
PrinterPaperInputTray               ::= SET OF PrinterPaperInputTrayStatus

PrinterPaperInputTrayStatus         ::= SEQUENCE
{
    printPaperInputSelect               [0] PrintPaperInputSelect,
    paperSize                           [1] PaperSize,
    paperDirection                      [2] PaperDirection,
    paperExistence                      [3] BOOLEAN             OPTIONAL
                                            -- If TRUE, input tray is not empty
                                            -- If FALSE, input tray is empty
}

FreeStorageSize                     ::= INTEGER
```

```
ListExcerptPrintJob              ::= SET OF ExcerptPrintJobDescription

ExcerptPrintJobDescription       ::= SEQUENCE
{
    jobHandle                        [0] JobHandle,
    jobStatusCode                    [1] JobStatusCode,
    printPriority                    [2] SimpleJobPriority
}
```

2.2.3.4.Job Related Operations

[Print] FU does not support job entry operation. Job related operations allow a client application to control a job execution such as canceling a job, and to know the job execution status. The usage of those commands and responses are described in "**Job-Related Messages**" section on page 33.

2.2.3.4.1.Controlling Job execution

[Print] Functional Unit defines printPriority attribute as **Job Control Attribute**. The following commands may be used to change the value of the attribute or cancel a job.

- [Print] FU Mandatory support common command
  - CancelJob
  - FreeJobHandle
  - ChangeJobAttribute
  - ACK and NACK

2.2.3.4.2.Job Status Notification

[Print] Functional Unit provides flexible ways for a client to know the status or the result of Print Service request. Refer to "**Job Status Notification**" section on page 34.

The following commands and responses are used for job status notification.

- [Print] FU Mandatory support command
  - QueryJobStatus
  - ACK and NACK
- [Print] FU Optional support common command
  - NotifyJobStatus

  Note) NotifyJobStatus, StartMonitorJobStatus and CancelMonitorJobStatus belong to the same Optional command Group, so an FU must support all these commands if it supports them.

2.2.3.4.3.Job Suspend/Resume

[Print] Functional Unit supports the following commands to suspend/resume jobs submitted by “Print” command.

- [Print] FU Mandatory support common command

  - SuspendJob
  - ResumeJob

2.2.3.4.4.Job Status Monitor Start/Cancel

[Print] Functional Unit supports the following commands to start/cancel job-status-monitoring.

- [Print] FU Optional support common command
  - StartMonitorJobStatus
  - CancelMonitorJobStatus

  Note) NotifyJobStatus, StartMonitorJobStatus and CancelMonitorJobStatus belong to the same Optional command Group, so an FU must support all these commands if it supports them.

2.2.3.4.5.List FU Job Status

[Print] Functional Unit supports the following commands to get the list of job in the [Print] Functional Unit.

- [Print] FU Mandatory support command
  - ListPrintJob
  - ACK and NACK

ListPrintJob Command

ListPrintJob command is used to get the list of job in the [Print] Functional Unit.

ASN.1 Syntax Definition

```
ListPrintJob                  ::= [APPLICATION tagListPrintJob] SEQUENCE
{
                                 COMPONENTS OF MsgHeader
}
```

Data transferred by TransferDataBlock command is as follows;

```
PrintJobList                  ::= SET OF PrintJobDescription
```

```
PrintJobDescription            ::= SEQUENCE
{
   jobHandle                      [0] JobHandle,
   requesterUserId                [1] UserID,
                                     -- "UserID" is set from the "UserID" specified in the Open
                                     -- Service request that has established a service session.
                                     -- Therefore, a client application must have registered as a
                                     -- [Client] FU to actually specify its "User ID" value so that it
                                     -- appears in the JobList.
   jobStatusCode                  [2] JobStatusCode,
   dataSize                       [3] INTEGER            OPTIONAL,
   queuedTime                     [4] DisplayString OPTIONAL,
   printPriority                  [5] SimpleJobPriority  OPTIONAL,
   printCopyCount                 [6] INTEGER            OPTIONAL,
   printPageCount                 [7] INTEGER            OPTIONAL,
   printFileName                  [8] DisplayString OPTIONAL
}
```

2.2.3.4.6.Job-Specific Reason Code

The following job-specific reason codes are defined to indicate the specific error conditions. The reason code supplements JobStatusCode that represents overall job status, e.g., completed, queued, suspended, and is an optional parameter present only when JobStatusCode is suspended or error. Refer to "**Data Type Definition**" on page 198 for the complete definition of the JobStatusCode.

The reason codes will be returned in a NotifyJobStatus or an ACK response to QueryJobStatus Command.

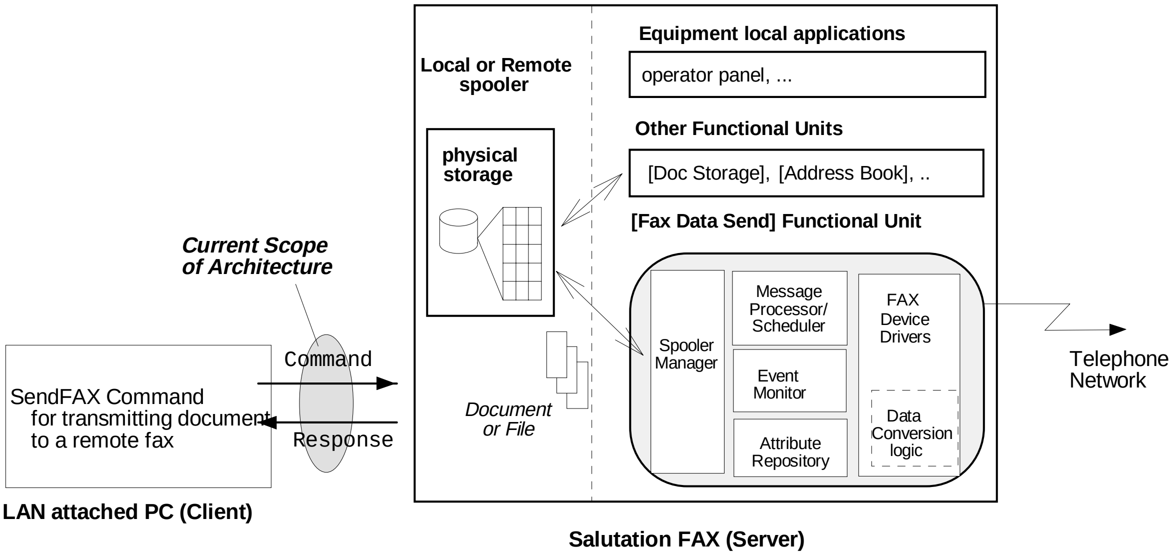
Name	**Description**	**ReasonCode**
suspendedByClientRequest	suspended by SuspendJob command	128
temporaryBusy	suspended due to equipment temporary busy.	129
waitingForRetry	in waiting mode for retry.	130
retryOut	terminated due to retry out of printing attempts.	131
printerError	terminated due to equipment detected errors, e.g., noPaper, noToner, and etc..	132

2.3.[FAX Data Send] Functional Unit

2.3.1.Overview

[FAX Data Send] Functional Unit provides a service for someone to request facsimile data transmission from a local or a remote equipment (including computer).

The following figure illustrates a configuration model to understand how **[FAX Data Send]** Functional Unit works with other resources within equipment and remote clients who issue service requests.



[FAX Data Send] provides a remote user with the capability to send a document or data via FAX protocol over telephone network, and to inquire the various status of the equipment.

[FAX Data Send] Functional Unit is considered to be composed of the following logical sub components or service elements.

- Message Processor and Scheduler for SendFAX command process
- Spooler Manager
- Event Monitor
- Attribute Repository
- FAX Device Drivers including for Fax control that may include Data conversion logic

2.3.2.List of Functional Unit Attributes

The following table describes the attributes defined for [FAX Data Send] Functional Unit, and specifies what protocol data unit will use those attributes.

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private/ Job Attribute
personalityProtocol	12000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No/No
supportedCommand	12001	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No/No
dynamicStatusId	12002	N/A	SET OF DynamicStatusID (setIntDoesContain)	No	No/No
numOfCalledSubscribers	12003	N/A	NumOfCalledSubscribers -- max integer value (intGreaterThanOrEqualTo)	No	No/No
spoolStorage	12004	N/A	SpoolStorage (boolEqualTo)	No	No/No
faxSendOrdering	12005	N/A (TelephoneNumberString be always specified when used)	FaxSendOrdering (boolEqualTo)	No	No/No
minimumCheckInterval -- the minimum allowed -- value to be set in the -- checkInterval parameter -- of a SubscribeEvent -- command	12006	N/A	INTEGER (intGreaterThanOrEqualTo)	No	No/No
documentFormat	12010	DataFormat	SET OF DataFormat (setIntDoesContain)	No	No/No
imageCompAlgorithm	12011	ImageCompAlgorithm	SET OF ImageCompAlgorithm (setIntDoesContain)	No	No/No
imageByteFillOrder	12012	ByteFillOrder	SET OF ByteFillOrder (setIntDoesContain)	No	No/No
imageResolution	12013	ImageResolution	SET OF ImageResolution (setIntDoesContain)	No	No/No
coverSheetGen	12020	CoverSheetGen	CoverSheetGen (boolEqualTo)	Yes	No/No
pageHeaderGen	12021	PageHeaderGen	PageHeaderGen (boolEqualTo)	Yes	No/No
faxProtocol	12030	FAXProtocol	SET OF FAXProtocol (setIntDoesContain)	Yes	No/No
requestPriority	12031	SimpleJobPriority (normal)	SET OF SimpleJobPriority (setIntDoesContain)	Yes	No/Yes

retryCount	12032	INTEGER	INTEGER (intGreaterThanOrEqualTo)	Yes	No/Yes
modeOfDataTransfer[3]	12035	DataTransferMode	SET OF DataTransferMode (setIntDoesContain)	Yes	No/No
dataLocationScheme	12036	N/A	SET OF DataLocationScheme (setIntDoesContain)	No	No/No
dataTransferTimeOutSettable	12037	N/A	BOOLEAN (boolEqualTo)	No	No/No
dataTransferTimeOutLength -- length in seconds for the FU -- to wait for the next message -- during a data transfer -- message sequence before -- detecting time-out exception	12038	INTEGER (N/A, if the previous dataTransferTimeOutSettable attribute is FALSE) —Global attribute indicates the —default length. If the global —attribute value is zero, the —default length is not fixed or —unknown. -- If the private attribute value is —set to zero, the FU should —wait as long as possible. —However, use of zero should —be avoided.	INTEGER (intGreaterThanOrEqualTo) -- if 0, not fixed or unknown -- (use of 0 should be avoided)	Yes (No, if the previous attribute is FALSE)	Yes/No (No, if the previous attribute is FALSE)

2.3.3.Message & Protocol

This section describes Service Request protocol for [FAX Data Send] when the Salutation Personality Protocol is selected.

2.3.3.1.Document Data Transfer Request

The following commands and responses are used for the Document Data Transfer Request procedure to send Fax data. Abstract syntax definition of common protocol data unit and its usage for document transfer procedure are described in either "**Data Transfer Messages**" section on page 18 or "

Document Transfer Procedure" section on page 69.

- [Fax Data Send] FU Mandatory support command
  - SendFAX
- [Fax Data Send] FU Mandatory support command
  - RequestDataTransfer
  - DataBlockDescription
  - TransferDataBlock
  - RequestNextData
  - ACK and NACK

[3] When “spoolStorage” = FALSE, only “delayed” mode is allowed for this attribute.

- [Fax Data Send] FU Optional support command
  - □ VendorEscape

2.3.3.1.1.Send Fax Data Request

SendFAX command is used to request [FAX Data Send] Functional Unit to send a data to a certain destinations via FAX protocol.

ASN.1 Syntax Definition

```
SendFAX                      ::= [APPLICATION tagSendFAX] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    modeOfDataTransfer           [0] DataTransferMode            OPTIONAL,
                                     -- Override Global / Private Attribute
    dataSource                   [1] DataLocation                DEFAULT client,
    dataHandle                   [2] DataHandle                  OPTIONAL,
                                     -- Omitted in immediate mode data transfer from client, or
                                     -- if the source data location is specified by URL
    inputDocumentFormat          [3] DocumentDataDescriptor      OPTIONAL,
                                     -- Present if and only if dataSource = url
    life                         [4] Life                        DEFAULT job,
                                     -- Specify how long FU should keep a job status:
                                     -- for job life or for session life or persistently.
    jobStatusNotificationMode    [5] JobStatusNotificationMode   OPTIONAL,
                                     -- If omitted, no notification is made.
    notificationScheme           [6] NotificationScheme          OPTIONAL,
                                     -- Omitted unless the job status notifications are to be
                                     -- sent to a [Client] FU other than the client that is
                                     -- sending this command
    faxControlAttribute          [7] FaxControlAttribute
}
```

Data Type used as SendFAX Command Parameters

```
FaxControlAttribute          ::= SEQUENCE
{
    tsInfo                       [0] TSInfo                      OPTIONAL,
    csInfoGroup                  [1] SET OF CSInfo,
    requestPriority              [2] SimpleJobPriority           OPTIONAL,
    retryCount                   [3] INTEGER                     OPTIONAL
}
```

```
TSInfo                  ::= SEQUENCE
{
    name                    [0] DisplayString           OPTIONAL,
    section                 [1] DisplayString           OPTIONAL,
    company                 [2] DisplayString           OPTIONAL,
    phoneNumber             [3] TelephoneNumberString           OPTIONAL,
    faxNumber               [4] TelephoneNumberString           OPTIONAL,
    address                 [5] DisplayString           OPTIONAL,
    subject                     [6] DisplayString                   OPTIONAL,
    coverSheetGen           [7] CoverSheetGen                   OPTIONAL,
    memoForCover            [8] DisplayString           OPTIONAL,
    pageHeaderGen           [9] PageHeaderGen                   OPTIONAL,
    memoForHeader           [10] DisplayString                  OPTIONAL
}

CoverSheetGen           ::= BOOLEAN

PageHeaderGen           ::= BOOLEAN

CSInfo                  ::= SEQUENCE
{
    jobEntryId              [0] JobEntryID,
    faxNumber               [1] TelephoneNumberString,
    subAddressNumber        [2] DisplayString           OPTIONAL,
    name                    [3] DisplayString           OPTIONAL,
    section                 [4] DisplayString           OPTIONAL,
    company                 [5] DisplayString           OPTIONAL,
    phoneNumber             [6] TelephoneNumberString           OPTIONAL,
    address                 [7] DisplayString           OPTIONAL,
    faxProtocol             [8] FAXProtocol                     DEFAULT g3,
    orderingData            [9] TelephoneNumberString           OPTIONAL
                                -- Ordering data is sent out by DTMF(Dual Tone Multi-
                                -- Frequency, G3) prior to G3 communication or UUI
                                -- (User User Information, G4) in G4 communication.
}

FAXProtocol             ::= ENUMERATED      -- G3 is assumed when omitted.
{
    g3                      (1),
    g4                      (2),
    auto                    (3)             -- Automatic selection of FaxProtocol to be used
}

FaxSendOrdering         ::= BOOLEAN
                                -- FaxSendOrdering is used in, for example, facsimile network
                                -- and mail service. "phoneNumber" specifies a phone
                                -- number for a service and orderingData is for final recipient
                                -- number
```

ACK Response

Parameter Name	Data Type	Note
parameter1	JobHandle	

NACK Response

Name	Description	ReturnCode
rcInvalidModeOfDataTransfer	modeOfDataTransfer is incorrect or not supported	128
rcInvalidDataSource	dataSource is incorrect or not supported	129
rcInvalidDataHandle	dataHandle is incorrect	130
rcInvalidInputDocumentFormat	inputDocumentFormat is incorrect or not supported	131
rcInvalidLife	life is incorrect or not supported	132
rcInvalidJobStatusNotificationMode	jobStatusNotificationMode is incorrect or not supported	133
rcInvalidNotificationScheme	notificationScheme is incorrect or not supported	134
rcInvalidCoverSheetGen	coverSheetGen is incorrect or not supported	135
rcInvalidPageHeaderGen	pageHeaderGen is incorrect or not supported	136
rcTooManyCalledSubscribers	The number of called subscribers exceeds the limit	137
rcInvalidFaxNumber	faxNumber is incorrect	138
rcInvalidSubAddressNumber	subAddressNumber is incorrect	139
rcInvalidFaxProtocol	faxProtocol is incorrect or not supported	140
rcInvalidOrderingData	orderingData is incorrect or not supported	141
rcInvalidRequestPriority	requestPriority is incorrect or not supported	142
rcInvalidRetryCount	retryCount is incorrect or not supported	143

Sample protocol sequences are provided below.

Example Protocol Sequence (1)

Client	Server

SendFAX(..., modeOfDataTransfer=delayed, dataSource=client, DataHandle, ...) =>

<= ACK(JobHandle)

COMMAND is enqueued in the Job Queue.

:

COMMAND is dequeued from the Job Queue.

--- Data Transfer Message Sequence Start ---

<= RequestDataTransfer(DataHandle)

DataBlockDescription(...) =>

<= RequestNextData

TransferDataBlock(Begin, End, Last) =>

<= ACK(NULL)

--- Data Transfer Message Sequence End ---

Client	Server

--- Data Transfer Message Sequence Start ---

SendFAX(..., modeOfDataTransfer=immediate, dataSource=client, ...) =>

<= RequestDataTransfer()

DataBlockDescription(...) =>

<= RequestNextData

TransferDataBlock(Begin, End, Last) =>

<= ACK(JobHandle)

--- Data Transfer Message Sequence End ---

COMMAND is enqueued in the Job Queue.

Client	Server

SendFAX(..., dataSource=functionalUnit, DataHandle, ...) =>

<= ACK(JobHandle)

Example Protocol Sequence (2)

Client	Server

SendFAX(..., jobStatusNotificationMode={{completed, error}, FALSE}, ...) =>

<= ACK(JobHandle)

:

<= NotifyJobStatus(JobHandle, completed)

ACK(NULL) =>

Client	Server

SendFAX(..., jobStatusNotificationMode={{completed, error}, TRUE}, ...) =>

<= ACK(JobHandle)

:

<= NotifyJobEntryStatus(JobHandle, JobEntryID, completed)

ACK(NULL) =>

:

<= NotifyJobEntryStatus(JobHandle, JobEntryID, completed)

ACK(NULL) =>

:

Client	Server

SendFAX(..., jobStatusNotificationMode={{}}, ...) =>

<= ACK(JobHandle)

:

QueryJobStatus(JobHandle) =>

<= ACK(completed)

Example Protocol Sequence (3)

Client	Server

SendFAX(...) =>

<= NACK(ReturnCode)

Example Protocol Sequence (4)

Client	**Server**
SendFAX(...) =>	
	<= ACK(JobHandle)
:	
ChangeJobAttribute(JobHandle, ...) =>	
	<= ACK(NULL)

Client	**Server**
SendFAX(...) =>	
	<= ACK(JobHandle)
:	
ChangeJobEntryAttribute(JobHandle, JobEntryID, ...) =>	
	<= ACK(NULL)

Example Protocol Sequence (5)

Client	**Server**
SendFAX(...) =>	
	<= ACK(JobHandle)
:	
CancelJob(JobHandle, ...) =>	
	<= ACK(NULL)

Client	**Server**
SendFAX(...) =>	
	<= ACK(JobHandle)
:	
CancelJobEntry(JobHandle, JobEntryID, ...) =>	
	<= ACK(NULL)

2.3.3.2.Attribute Operations

The following command and response are used for attribute controls. The usage of those commands and responses are described in "**Attribute Repository Messages**" section on page 30.

- [Fax Data Send] FU Mandatory support common command
  - □ GetPrivateAttribute
  - □ GetGlobalAttribute

- □ SetPrivateAttribute
- □ ACK and NACK

Attributes affected by the above commands are listed in "List of Functional Unit Attributes" section.

2.3.3.3.Dynamic Status Operations

Dynamic Status operations allow a client to know the aspect of Functional Unit and the transition in the aspects. **Dynamic Status Parameter** describes the aspects. A client may query the current values of Dynamic Status Parameter, and request [Fax Data Send] to notify an Event when any transition occurs in the values of Dynamic Status Parameter.

The following commands and response are used for dynamic status operations. The usage of those commands and responses are described in "**Dynamic Status Messages**" section on page 49.

- [Fax Data Send] FU Mandatory support common command
  - □ QueryDynamicStatus
  - □ ACK and NACK
- [Fax Data Send] FU Optional support common command
  - □ SubscribeEvent, UnsubscribeEvent and NotifyEvent (These commands belong to the same Optional group, so an FU must support all these commands if it supports them.)

The following Dynamic Status Parameters are defined for [Fax Data Send] Functional Unit.

Dynamic Status Parameter	**Query**	**Event**	**ID**	**Description**
FaxSendStatus	Yes	Yes	12000	status of FAX equipment at sending side.
FaxSendFreeStorageSize	Yes	No	12001	storage size available for spool.
FaxSendErrorStatus	Yes	No	12002	the detail error status information.

Data Type of Dynamic Status Parameter

```
FaxSendStatus                 ::= ENUMERATED
{
    powerFailure                  (0),
    warmingUp                     (1),
    offline                       (2),
    ready                         (3),
    sending                       (4),
    receiving                     (5),
    error                         (6),
    others                        (127)
}

FaxSendFreeStorageSize        ::= INTEGER
```

```
FaxSendErrorStatus                ::= SEQUENCE
{
    systemError                     [0] DisplayString,          -- detail description
    others                          [1] DisplayString-- detail description
}
```

2.3.3.4.Job Related Operations

[Fax Data Send] FU supports job entry operation since multiple destinations can be specified in single SendFAX command. A client application can control the way of executing a job or job entry, and also know the status of the job or job entry execution. The usage of those commands and responses are described in "**Job-Related Messages**" section on page 33.

[Fax Data Send] FU specific job status transition is as follows:

When data transfer from a client to sending [Fax Data Send] FU is completed, the job status becomes “Queued”.

When one of job entry is started, to be sent to the receiving Fax, the job status becomes “Started”, and when ALL job entries are sent, the job status becomes “Completed”.

2.3.3.4.1.Controlling Job execution

[Fax Data Send] Functional Unit defines requestPriority and retryCount attribute as **Job Control Attributes**. The following commands may be used to change the value of the attributes or cancel a job or job entry.

- [Fax Data Send] FU Mandatory support common command
  - □ CancelJob
  - □ FreeJobHandle
  - □ ChangeJobAttribute and
  - □ ACK and NACK
- [Fax Data Send] FU Optional support common command
  - □ CancelJobEntry
  - □ ChangeJobEntryAttribute

Note) CancelJobEntry, ChangeJobEntryAttribute, QueryJobEntryStatus, NotifyJobEntryStatus, SuspendJobEntry, and ResumeJobEntry belong to the same Optional command Group, so an FU must support all these commands if it supports them.

2.3.3.4.2.Job Status Notification

[FAX Data Send] Functional Unit provides flexible ways for a client to know the status or the result of Print Service request. Refer to "**Job Status Notification**" section on page 34.

The following commands and responses are used for job status notification.

- [Fax Data Send] FU Mandatory support common command
  - □ QueryJobStatus
  - □ ACK and NACK

- [Fax Data Send] FU Optional support common command
  - □ NotifyJobStatus

    Note) NotifyJobStatus, StartMonitorJobStatus and CancelMonitorJobStatus belong to the same Optional command Group, so an FU must support all these commands if it supports them.

  - □ QueryJobEntryStatus
  - □ NotifyJobEntryStatus

    Note) CancelJobEntry, ChangeJobEntryAttribute, QueryJobEntryStatus, NotifyJobEntryStatus, SuspendJobEntry, and ResumeJobEntry belong to the same Optional command Group, so an FU must support all these commands if it supports them.

2.3.3.4.3.Job Suspend/Resume

[Fax Data Send] Functional Unit supports the following commands to suspend/resume jobs submitted by “SendFAX” command.

- [Fax Data Send] FU Mandatory support common command
  - □ SuspendJob
  - □ ResumeJob
- [Fax Data Send] FU Optional support common command
  - □ SuspendJobEntry
  - □ ResumeJobEntry

    Note) CancelJobEntry, ChangeJobEntryAttribute, QueryJobEntryStatus, NotifyJobEntryStatus, SuspendJobEntry, and ResumeJobEntry belong to the same Optional command Group, so an FU must support all these commands if it supports them.

2.3.3.4.4.Job Status Monitor Start/Cancel

[Fax Data Send] Functional Unit supports the following commands to start/cancel job-status-monitoring.

- [Fax Data Send] FU Optional support common command
  - □ StartMonitorJobStatus
  - □ CancelMonitorJobStatus

    Note) NotifyJobStatus, StartMonitorJobStatus and CancelMonitorJobStatus belong to the same Optional command Group, so an FU must support all these commands if it supports them.

2.3.3.4.5.List FU Job Status

- [Fax Data Send] FU Mandatory support command
  - □ ListFaxJob
  - □ ACK and NACK

ListFaxJob Command

ListFaxJob command is used to get the list of job in the **[Fax Data Send]** Functional Unit.

ASN.1 Syntax Definition

```
ListFaxtJob                     ::= [APPLICATION tagListFaxtJob] SEQUENCE
{
                                    COMPONENTS OF MsgHeader
}
```

Data transferred by TransferDataBlock command is as follows.

```
FaxJobList                      ::= SET OF FaxJobDescription

FaxJobDescription               ::= SEQUENCE
{
    jobHandle                       [0] JobHandle,
    requesterUserId                 [1] UserID,
                                        -- "UserID" is set from the "UserID" specified in the Open Service
                                        -- request that has established a service session. Therefore, a client
                                        -- application must have registered as a [Client] FU to actually
                                        -- specify its "User ID" value so that it appears in the JobList.
    jobStatusCode                   [2] JobStatusCode,
    dataSize                        [3] INTEGER              OPTIONAL,
    numOfJobEntries                 [4] INTEGER              OPTIONAL
}
```

2.3.3.4.6.Job-Specific Reason Code

The following job-specific reason codes are defined to indicate the specific error conditions. The reason code supplements JobStatusCode that represents overall job status, e.g., completed, queued, suspended, and is an optional parameter present only when JobStatusCode is suspended or error. Refer to "**Data Type Definition**" on page 198 for the complete definition of the JobStatusCode.

The reason codes will be returned in a NotifyJobStatus, a NotifyJobEntryStatus, or an ACK response to QueryJobStatus or QueryJobEntryStatus Command.

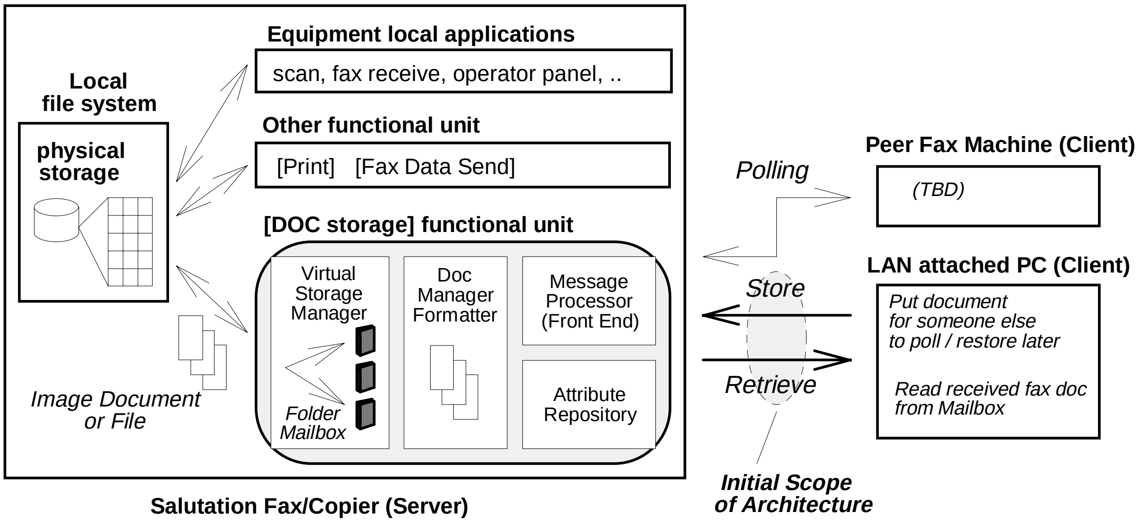
Name	Description	ReasonCode
timeOut	time-out detected during get-line. (When zero is specified in retryCount)	128
retryOut	terminated due to retry out. (When zero is specified for retryCount, this parameter is not returned. Instead calledSubscriberBusy or timeOut is returned,)	129
calledSubscriberBusy	busy status detected for called subscriber.	130
modemShiftDownFailed	connection failed with the lowest speed.	131
callSetUpFailed	call setup failed.	132
negotiationFailed	negotiation failed.	133
notReceiveExpectedFrame	expecting frame(s) not received on G3 protocol.	134
receiveUnexpectedFrame	unexpected frame(s) received on G3 protocol.	135
thirdTryFail	retried-out during G3 protocol.	136
waitingForRetry	in waiting mode for retry.	137

2.4.[DOC Storage] Functional Unit

2.4.1.Overview

[DOC Storage] Functional Unit is used as a temporary spooling storage for document data (handled by scanning, printing and Faxing operation) and non-document data like file data (handled as application data or executable code). It provides a client with simple access methods to temporary spooling storage, and defines minimum functions for the purpose. [DOC Storage] Functional Unit abstracts a container of document and file data as folder, and may contain more than one folders. Implementation of [DOC Storage] Functional Unit can be typically found in a facsimile machine, copier, printing equipment, or for storage of file data like device drivers, application data, and so on.

The following figure illustrates a configuration model to understand how [DOC Storage] Functional Unit works with other resources within equipment and with remote clients who issue service requests.


Equipment local applications
scan, fax receive, operator panel, ..
Local file system
physical storage
Other functional unit
[Print] [Fax Data Send]
[DOC storage] functional unit
Virtual Storage Manager
Doc Manager Formatter
Message Processor (Front End)
Attribute Repository
Folder Mailbox
Image Document or File
Salutation Fax/Copier (Server)
Polling
Store
Retrieve
Initial Scope of Architecture
Peer Fax Machine (Client)
(TBD)
LAN attached PC (Client)
Put document for someone else to poll / restore later
Read received fax doc from Mailbox

[DOC Storage] Functional Unit provides the interface for a remote client to store and retrieve document and file data. Data source in "document storing" service and data destination in "document retrieval" service are "client applications" by default. The data source/destination can be other than clients when the data source/destination attributes are explicitly set.

[DOC Storage] Functional Unit may share incoming/outgoing image document and/or file data with other Functional Units or local applications within equipment via a local file system. Client applications can request document/file data exchange among those local Functional Units by using the procedure defined in "Document Systems Overview" section. However, it is not the scope of this architecture to define the mechanism how [DOC Storage] interfaces with the local file system. Some examples of interworking between and other Functional Units/applications are :

- scanning document → [DOC Storage] → PC, FAX, Printing equipment
- received document at FAX → [DOC Storage] → PC, FAX, Printing equipment
- generated document in PC → [DOC Storage] → FAX, Printing equipment
- file data → [DOC Storage] → [Client] FU

[DOC Storage] Functional Unit is considered to be composed of the following logical sub-components or service elements.

- Front-end message processor
- Attribute Repository manager
- Document manager/formatter
- Virtual storage manager (directory manager)

2.4.2.List of Functional Unit Attributes

The following table describes the attributes defined for [DOC Storage] Functional Unit, and specifies what protocol data unit will use those attributes.

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private Attribute
personalityProtocol	11000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No
supportedCommand	11001	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No
dynamicStatusId	11002	N/A	SET OF DynamicStatusID (setIntDoesContain)	No	No
readWriteCapability	11003	AccessMode	SET OF AccessMode (setIntDoesContain)	No	No
minimumCheckInterval -- the minimum allowed -- value to be set in the —checkInterval parameter -- of a SubscribeEvent —command	11004	N/A	INTEGER (intGreaterThanOrEqualTo)	No	No
typeOfContent	11009	DataContent	SET OF DataContent (setIntDoesContain)	No	No
modeOfStore	11010	DataStoreMode	SET OF DataStoreMode (setIntDoesContain)	Yes	Yes
documentFormat	11011	DataFormat	SET OF DataFormat (setIntDoesContain)	No	No
imageCompAlgorithm	11012	ImageCompAlgorithm	SET OF ImageCompAlgorithm (setIntDoesContain)	No	No
imageByteFillOrder	11013	ByteFillOrder	SET OF ByteFillOrder (setIntDoesContain)	No	No
imageResolution	11014	ImageResolution	SET OF ImageResolution (setIntDoesContain)	No	No
dataLocationScheme	11030	N/A	SET OF DataLocationScheme (setIntDoesContain)	No	No
dataTransferTimeOutSettable	11031	N/A	BOOLEAN (boolEqualTo)	No	No

dataTransferTimeOutLength -- length in seconds for the FU -- to wait for the next message -- during a data transfer -- message sequence before -- detecting time-out exception	11032	INTEGER (N/A, if the previous dataTransferTimeOutSettable attribute is FALSE) -- Global attribute indicates the -- default length. If the global -- attribute value is zero, the -- default length is not fixed or -- unknown. -- If the private attribute value is -- set to zero, the FU should -- wait as long as possible. -- However, use of zero should -- be avoided.	INTEGER (intGreaterThanOrEqualTo) -- if 0, not fixed or unknown -- (use of 0 should be avoided)	Yes (No, if the previous attribute is FALSE)	Yes (No, if the previous attribute is FALSE)

“typeOfContent” determines the content of the stored data in the [DOC Storage] Functional Unit, i.e. document or file data. If [DOC Storage] Functional Unit supports the file data type, it also should support the fileMode of “modeOfStore”, so that the data is transparent for a client or a [DOC Storage] Functional Unit.

"modeOfStore" determines the modes of operation on data, i.e., documentDataMode (non-transparent) and fileMode (transparent). In non-transparent mode the document contents are interpreted by a client and [DOC Storage] Functional Unit, and a data block boundary corresponds to a page boundary in transmission. In transparent mode a block boundary if any does not represent a page or meaningful boundary, and [DOC Storage] Functional Unit transmits a whole data in a block.

Document related attributes must be distinguished from the document data descriptor associated with document data. "documentFormat" attribute configures how [DOC Storage] Functional Unit should handle a stored document. It is meaningful only when “dataDescriptor” in DataBlockDescription is set to document .

When "modeOfStore" attribute is set to document mode (non-transparent mode), [DOC Storage] Functional Unit refers to "documentFormat" attribute and becomes conscious of the stored document format. For example, when "documentFormat" attribute is set to bi-level image stream and "modeOfStore" attribute is set to document data mode (non-transparent mode), a stored document is treated as a sequence of pages. When "modeOfStore" attribute is set to file mode (transparent mode), [DOC Storage] Functional Unit becomes unconscious of the content of data and does not refer document related attributes, and transmits the whole data to a client in a block rather than multiple blocks.

2.4.3.Message & Protocol

This section describes service request protocol for [DOC Storage] Functional Unit under Salutation Personality Protocol.

2.4.3.1.Document Control and Data Transfer Request

The following request procedures are defined for [DOC Storage] Functional Unit.

□ Document Retrieval Request

□ Document Storing Request

- □ Document Copying and Moving Request
- □ Folder Creation Request
- □ Folder Deletion Request
- □ Folder Listing Request
- □ Document Listing Request
- □ Folder Descriptions Updating Request
- □ Document Descriptions Updating Request

The following commands and responses are used for the Document Control and Data Transfer Request procedure to access a document. Abstract syntax definition of common protocol data unit and its usage for document transfer procedure are described in either "**Data Transfer Messages**" section on page 18 or " Document Transfer Procedure" section on page 69.

- [DOC Storage] FU Mandatory support Command
  - □ RetrieveDoc
  - □ StoreDoc
  - □ ListFolder
  - □ ListFolderDoc
- [DOC Storage] FU Optional support Command
  - □ DeleteDoc, CopyDoc, MoveDoc, ChangeDocDesc, CreateFolder, DeleteFolder, and ChangeFolderDesc (These commands belong to the same optional group, so an FU must support all these commands if it supports.)
- [DOC Storage] FU Mandatory support common Command
  - □ RequestDataTransfer
  - □ DataBlockDescription
  - □ TransferDataBlock
  - □ RequestNextData
  - □ ACK and NACK
- [DOC Storage] FU Optional support Command
  - □ VendorEscape

Sample protocol sequences for each command request procedures and abstract syntax definition of each command are provided in each chapter.

2.4.3.1.1.Document Retrieval Request

Example Protocol Sequence (1)

Client	**Server**
RetrieveDoc(..., dataDestination=client) =>	
	<= DataBlockDescription
RequestNextData =>	
	<= TransferDataBlock
ACK(NULL) =>	

Example Protocol Sequence (2)

Client	**Server**
RetrieveDoc(..., dataDestination=exportPool) =>	
	<= ACK(DataHandle)

Example Protocol Sequence (3)

Client	**Server**
RetrieveDoc(...) =>	
	<= NACK(ReturnCode)

RetrieveDoc Command

RetrieveDoc command is used to retrieve a document from a certain folder managed by [DOC Storage] Functional Unit. Prior to this request, client may get information of the target document and the folder containing it by executing "List Document" command request procedure.

By default, documents are transferred to the client by using document transfer procedure described in "Document Systems Overview" section. When data destination parameter is set to the Export Pool, the document is prepared for access by other Functional Unit, and export data handle is returned in ACK response.

ASN.1 Syntax Definition

```
RetrieveDoc                   ::= [APPLICATION tagRetrieveDoc] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    folderId                     [0] FolderID,
    documentId                   [1] DocumentID,
    dataDestination              [2] DataLocation         DEFAULT client,
    startDataBlock               [3] INTEGER              DEFAULT 1,
                                     -- If omitted, the document is retrieved
                                     -- from the first data block.
    endDataBlock                 [4] INTEGER              OPTIONAL
                                     -- If omitted, the document is retrieved
                                     -- through the last data block.
}
```

ACK Response

Indicates that RetrieveDoc command request is successfully processed. If Export Pool is specified as the data destination, Export DataHandle is returned.

Parameter Name	Data Type	Note
parameter1	DataHandle	Optional

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcAccessRejected	access is not authorized for the user	130
rcInvalidDataDestination	dataDestination is incorrect or not supported	131
rcInvalidStartDataBlock	startDataBlock is incorrect	132
rcInvalidEndDataBlock	endDataBlock is incorrect	133

2.4.3.1.2.Document Storing Request

Example Protocol Sequence (1)

Client	**Server**
StoreDoc(..., dataSource=client, ...) =>	
	<= RequestDataTransfer()
DataBlockDescription =>	
	<= RequestNextData
TransferDataBlock =>	
	<= ACK(DocumentID)

Example Protocol Sequence (2)

Client	**Server**
StoreDoc(DataHandle, ... , dataSource=functionalUnit, ...) =>	
	<= ACK(DocumentID)

Example Protocol Sequence (3)

Client	**Server**
StoreDoc(...) =>	
	<= NACK(ReturnCode)

StoreDoc Command

StoreDoc command is used to store a document into a certain folder managed by [DOC Storage] Functional Unit.

By default, documents are transferred from the client by using document transfer procedure described in "Document Systems Overview" section. When data source parameter is set to another Functional Unit, the document is transferred from the Export Pool of the specified Functional Unit before ACK response is returned to the client.

The [DOC Storage] Functional Unit stores the data transferred by the Data Transfer Message Sequence as follows:

- When the modeOfStore is “documentDataMode”

  The boundaries of the data blocks of transferred data are preserved. This mode is to be used in Non-Transparent Mode.

- When the modeOfStore is “fileMode”

  The data blocks of transferred data are merged into a single data block. This mode is to be used in Transparent Mode.

ASN.1 Syntax Definition

```
StoreDoc                     ::= [APPLICATION tagStoreDoc] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
   folderId                      [0] FolderID,
   dataSource                    [1] DataLocation              DEFAULT client,
   dataHandle                    [2] DataHandle                OPTIONAL,
                                    -- Exists only if dataSource = Functional Unit
   modeOfStore                   [3] DataStoreMode             OPTIONAL,
                                    -- Override Global / Private Attribute
   inputDocumentFormat           [4] DocumentDataDescriptor    OPTIONAL,
                                    -- Present if and only if dataSource = url
   ownerName                     [5] OwnerName                 OPTIONAL,
   docComment                    [6] DocComment                OPTIONAL,
   typeOfContent                 [7] DataContent               OPTIONAL
}
```

ACK Response

Indicates that StoreDoc command request is successfully processed. Document ID is returned.

Parameter Name	Data Type	Note
parameter1	DocumentID	

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown	128
rcAccessRejected	access is not authorized for the user	130
rcInvalidDataSource	dataSource is incorrect or not supported	131
rcInvalidDataHandle	dataHandle is unknown	132
rcInvalidModeOfStore	modeOfStore is incorrect or not supported	133
rcInvalidInputDocumentFormat	inputDocumentFormat is incorrect or not supported	134
rcStorageFull	storage is full	135
rcInvalidTypeOfContent	typeOfContent is incorrect or not supported	136

2.4.3.1.3.Document Deleting Request

Example Protocol Sequence (1)

Client	Server
DeleteDoc(...) =>	
	<= ACK(NULL)

Example Protocol Sequence (2)

Client	Server
DeleteDoc(...) =>	
	<= NACK(ReturnCode)

DeleteDoc Command

DeleteDoc command is used to explicitly remove a document.

ASN.1 Syntax Definition

```
DeleteDoc                    ::= [APPLICATION tagDeleteDoc] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
   folderId                      [0] FolderID,
   documentId                    [1] DocumentID
}
```

ACK Response

Indicates that DeleteDoc request is successfully processed. No parameter is returned.

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcAccessRejected	access is not authorized for the user	130

2.4.3.1.4.Document Copying Request

Example Protocol Sequence (1)

Client	Server

CopyDoc(...) =>

<= ACK(DocumentID)

Example Protocol Sequence (2)

Client	Server

CopyDoc(...) =>

<= NACK(ReturnCode)

CopyDoc Command

CopyDoc command is used to copy a document within a storage maintained by [DOC Storage] Functional Unit. This operation allows a client to copy a document without transferring a document by using RetrieveDoc and StoreDoc command.

ASN.1 Syntax Definition

```
CopyDoc                 ::= [APPLICATION tagCopyDoc] SEQUENCE
{
                            COMPONENTS OF MsgHeader,
   sourceFolder            [0] FolderID,
   documentId              [1] DocumentID,
   destinationFolder       [2] FolderID              OPTIONAL,
                           -- if omitted, the same as sourceFolder
   updateDateTime          [3] BOOLEAN               DEFAULT FALSE
                           -- if TRUE, update the document's
                           -- creationDateTime with the current time.
                           -- if FALSE or omitted, use the document's
                           -- old creationDateTime.
}
```

ACK Response

Indicates that CopyDoc command request is successfully processed. Document ID for a new document is returned.

Parameter Name	Data Type	Note
parameter1	DocumentID	

NACK Response

Name	Description	ReturnCode
rcInvalidSourceFolderId	sourceFolder is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcSourceAccessRejected	access to the source folder/document is not authorized for the user	130
rcInvalidDestinationFolderId	destinationFolder is unknown	131
rcDestinationAccessRejected	access to the destination folder is not authorized for the user	132

2.4.3.1.5.Document Moving Request

Example Protocol Sequence (1)

Client	Server
MoveDoc(...) =>	
	<= ACK(DocumentID)

Example Protocol Sequence (2)

Client	Server
MoveDoc(...) =>	
	<= NACK(ReturnCode)

MoveDoc Command

MoveDoc command is used to move a document to another folder within a storage maintained by [DOC Storage] Functional Unit. This operation allows a client to move a document without transferring a document by using RetrieveDoc, StoreDoc and DeleteDoc command.

ASN.1 Syntax Definition

```
MoveDoc                    ::= [APPLICATION tagMoveDoc] SEQUENCE
{
                               COMPONENTS OF MsgHeader,
   sourceFolder                [0] FolderID,
   documentId                  [1] DocumentID,
   destinationFolder           [2] FolderID              OPTIONAL,
                               -- if omitted, the same as sourceFolder
   updateDateTime              [3] BOOLEAN               DEFAULT FALSE
                               -- if TRUE, update the document's
                               -- creationDateTime with the current time.
                               -- if FALSE or omitted, use the document's
                               -- old creationDateTime.
}
```

ACK Response

Indicates that MoveDoc command request is successfully processed. Document ID for a new document is returned.

Parameter Name	Data Type	Note
parameter1	DocumentID	

NACK Response

Name	Description	ReturnCode
rcInvalidSourceFolderId	sourceFolder is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcSourceAccessRejected	access to the source folder/document is not authorized for the user	130
rcInvalidDestinationFolderId	destinationFolder is unknown	131
rcDestinationAccessRejected	access to the destination folder is not authorized for the user	132

2.4.3.1.6.Document Descriptions Updating Request

Example Protocol Sequence (1)

Client	Server
ChangeDocDesc(...) =>	
	<= ACK(NULL)

Example Protocol Sequence (2)

Client	**Server**

ChangeDocDesc(...) =>

<= NACK(ReturnCode)

ChangeDocDesc Command

ChangeDocDesc command is used to update the descriptions associated with a document.

ASN.1 Syntax Definition

```
ChangeDocDesc                 ::= [APPLICATION tagChangeDocDesc] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
    folderId                      [0] FolderID,
    documentId                    [1] DocumentID,
    ownerName                     [2] OwnerName          OPTIONAL,
    docComment                    [3] DocComment         OPTIONAL
}
```

ACK Response

Indicates that ChangeDocDesc request is successfully processed. No parameter is returned.

NACK Response

Name	**Description**	**ReturnCode**
rcInvalidFolderId	folderId is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcAccessRejected	access is not authorized for the user	130

2.4.3.1.7.Folder Creation Request

Example Protocol Sequence (1)

Client	Server

CreateFolder(...) =>

<= ACK(FolderID)

Example Protocol Sequence (2)

Client	Server

CreateFolder(...) =>

<= NACK(ReturnCode)

CreateFolder Command

CreateFolder command is used to create a folder in a storage maintained by [DOC Storage] Functional Unit.

ASN.1 Syntax Definition

```
CreateFolder                  ::= [APPLICATION tagCreateFolder] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
    ownerName                     [0] OwnerName              OPTIONAL,
    folderComment                 [1] FolderComment          OPTIONAL
}
```

ACK Response

Indicates that CreateFolder command request is successfully processed. FolderID for a new document is returned.

Parameter Name	Data Type	Note
parameter1	FolderID	

NACK Response

Indicates that CreateFolder command request is rejected. There is no message specific return code.

Default Public Folder

A folder of which “Folder ID” equals to 0 (Zero) is defined for special purpose folder, called **Default Public Folder**. Default Public Folder is useful when a client just needs to temporarily store a document into DOC Storage without restrictive access control. A client usually locate a folder prior to storing a document by issuing ListFolder or CreateFolder command. Default Public Folder allows a client to store a document without these steps, since it is well-known to the client. Default Public Folder has the following characteristics:

- There is no restrictive access control to the Default Public Folder.
- ChangeFolderDesc and DeleteFolder command cannot be used against the Default Public Folder.
- ownerName, creationDateTime and description of the Default Public Folder is implementation dependent.
- The provision of Default Public Folder by [DOC Storage] FU depends on FU implementation. If not implemented, an attempt to store a document to Default Public Folder is simply rejected with the return code of rcFolderNotFound.
- When a [DOC Storage] FU implementation does not provide Default Public Folder, it shall never return FolderID=0 in response to a CreateFolder command.

Example Protocol Sequences

Client	**Server**
-- *to use of ListFolderDoc and then StoreDoc with explicit Folder ID*	
ListFolderDoc(...) =>	
	<= **TransferDataBlock(a list of folders)**
-- *to find a folder to use*	
StoreDoc(...) =>	
-- *to use of CreateFolder and then StoreDoc with explicit Folder ID*	
CreateFolder(...) =>	
	<= **ACK(FolderID)**
StoreDoc(...) =>	
-- *to use of Default Public Folder*	
StoreDoc(., folderId=0, ...) =>	

2.4.3.1.8.Folder Description Updating Request

Example Protocol Sequence (1)

Client	**Server**

ChangeFolderDesc(...) =>

<= ACK(NULL)

Example Protocol Sequence (2)

Client	**Server**

ChangeFolderDesc(...) =>

<= NACK(ReturnCode)

ChangeFolderDesc Command

ChangeFolderDesc command is used to update the descriptions associated with a folder.

ASN.1 Syntax Definition

```
ChangeFolderDesc              ::= [APPLICATION tagChangeFolderDesc] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   folderId                       [0] FolderID,
   ownerName                      [1] OwnerName               OPTIONAL,
   folderComment                  [2] FolderComment           OPTIONAL
}
```

ACK Response

Indicates that command request is successfully processed. No parameter is returned.

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown or incorrect (0)	128
rcAccessRejected	access is not authorized for the user	130

2.4.3.1.9.Folder Deletion Request

Example Protocol Sequence (1)

Client	Server

```
DeleteFolder(...) =>
                                                  <= ACK(NULL)
```

Example Protocol Sequence (2)

Client	Server

```
DeleteFolder(...) =>
                                                  <= NACK(ReturnCode)
```

DeleteFolder Command

DeleteFolder command is used to delete a folder which does not hold any document.

ASN.1 Syntax Definition

```
DeleteFolder                ::= [APPLICATION tagDeleteFolder] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
folderId                        [0] FolderID
                                -- Folder should be empty before deleted.
}
```

ACK Response

Indicates that command request is successfully processed. No parameter is returned.

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown or incorrect (0)	128
rcAccessRejected	access is not authorized for the user	130
rcFolderNotEmpty	folder contains document(s)	131

2.4.3.1.10.Folder Listing Request

Example Protocol Sequence (1)

Client	**Server**
ListFolder(...) =>	
	<= TransferDataBlock
ACK(NULL) =>	

Example Protocol Sequence (2)

Client	**Server**
ListFolder(...)=>	
	<= NACK(ReturnCode)

ListFolder Command

ListFolder command is used to get a list of folders managed by [DOC Storage] Functional Unit.

The list is transferred from the [DOC Storage] Functional Unit to the client by using a Data Transfer Message Sequence as follows:

> The list of folders is transferred as “data” consisting of one data block which may be split into multiple data block segments. Each data block segment is of **FolderList** data type which is defined as **SET OF FolderDescription** as shown below. For example, if the [DOC Storage] Functional Unit contains 900 folders, the description of the first 300 folders may be sent in the first data block segment, that of the next 300 folders in the 2nd data block segment, and that of the last 300 folders in the last data block segment. Although each data block segment contains only a part of the whole folders set, the receiving application can decode (according to the BER) each data block segment without waiting for the next data block segment.

ASN.1 Syntax Definition

```
ListFolder                    ::= [APPLICATION tagListFolder] SEQUENCE
{
                                  COMPONENTS OF MsgHeader
}

FolderList                    ::= SET OF FolderDescription
```

```
FolderDescription                ::= SEQUENCE
{
    folderId                         [0] FolderID,
    ownerName                        [1] OwnerName                   OPTIONAL,
    folderComment                    [2] FolderComment               OPTIONAL,
    createDateTime                   [3] DisplayString          OPTIONAL,
    usedSize                         [4] INTEGER                     OPTIONAL,
                                         -- size in bytes occupied by the documents in this folder
    freeSize                         [5] INTEGER                     OPTIONAL,
                                         -- size in bytes of free area in this folder
    numberOfDocuments                [6] INTEGER                     OPTIONAL
}
```

ACK Response

Indicates that command request is successfully processed. No parameter is returned.

NACK Response

Indicates that ListFolder command request is rejected. There is no message specific return code.

2.4.3.1.11.Document Listing Request

Example Protocol Sequence (1)

Client	**Server**
ListFolderDoc(...) =>	
	<= TransferDataBlock
ACK(NULL) =>	

Example Protocol Sequence (2)

Client	**Server**
ListFolderDoc(...)=>	
	<= NACK(ReturnCode)

ListFolderDoc Command

ListFolderDoc command is used to get a list of documents stored in a certain folder managed by [DOC Storage] Functional Unit.

The list is transferred from the [DOC Storage] Functional Unit to the client by using a Data Transfer Message Sequence as follows:

> The list of documents is transferred as “data” consisting of one data block which may be split into multiple data block segments. Each data block segment is of **DocList** data type which is

defined as **SET OF DocDescription** as shown below. For example, if the specified folder of the [DOC Storage] Functional Unit contains 900 documents, the description of the first 300 documents may be sent in the first data block segment, that of the next 300 documents in the 2nd data block segment, and that of the last 300 documents in the last data block segment. Although each data block segment contains only a part of the whole documents set, the receiving application can decode (according to the BER) each data block segment without waiting for the next data block segment.

ASN.1 Syntax Definition

```
ListFolderDoc              ::= [APPLICATION tagListFolderDoc] SEQUENCE
{
                               COMPONENTS OF MsgHeader,
   folderId                    [0] FolderID
}

DocList                    ::= SET OF DocDescription

DocDescription             ::= SEQUENCE
{
   documentId                  [0] DocumentID,
   ownerName                   [1] OwnerName                 OPTIONAL,
   docComment                  [2] DocComment                OPTIONAL,
   createDateTime              [3] DisplayString             OPTIONAL,
   size                        [4] INTEGER                   OPTIONAL,
                                   -- size in bytes of this document
   numberOfBlocks              [5] INTEGER                   OPTIONAL,
                                   -- size in blocks that may be useful in RetrieveDoc to specify
                                   -- startDataBlock and endDataBlock parameter.
   typeOfContent               [6] DataContent               OPTIONAL
}
```

ACK Response

Indicates that command request is successfully processed. No parameter is returned.

NACK Response

Name	**Description**	**ReturnCode**
rcInvalidFolderId	folderId is unknown or incorrect (0)	128
rcAccessRejected	access is not authorized for the user	130

2.4.3.2.Attribute Operations

The following command and response are used for attribute control. The usage of those commands and responses are described in "**Attribute Repository Messages**" section on page 30.

- [DOC Storage] FU Mandatory support common Command
  - □ GetPrivateAttribute

- □ GetGlobalAttribute
- □ SetPrivateAttribute
- □ ACK and NACK

Attributes affected by the above commands are listed in "List of Functional Unit Attributes" section.

2.4.3.3.Dynamic Status Operations

Dynamic Status operations allow a client to know the aspect of Functional Unit and the transition in the aspects. **Dynamic Status Parameter** describes the aspects. A client may query the current values of Dynamic Status Parameter, and request [Fax Data Send] to notify an Event when any transition occurs in the values of Dynamic Status Parameter.

The following commands and response are used for dynamic status operations. The usage of those commands and responses are described in "**Dynamic Status Messages**" section on page 49.

- [DOC Storage] FU Mandatory support Command
  - □ QueryDynamicStatus
  - □ ACK and NACK
- [DOC Storage] FU Mandatory support Command
  - □ SubscribeEvent, UnsubscribeEvent, and NotifyEvent (These commands belong to the same Optional Group, so an FU must support all these commands if it supports them.)

The following Dynamic Status Parameter is defined for [DOC Storage] Functional Unit. A client may query the current values of Dynamic Status Parameter, or request [DOC Storage] to notify an Event when any transition occurs in the values of Dynamic Status Parameter.

Dynamic Status Parameter	Query	Event	ID	Description
FreeStorageSize	Yes	No	11000	available storage size.
OperatorIntervention	No	Yes	11001	a warning message to operator or administrator to request human intervention
OperatorInformation	No	Yes	11002	an informational message to operator or administrator

Data Type of Dynamic Status Parameter

```
FreeStorageSize              ::= INTEGER

OperatorIntervention         ::= SEQUENCE
{
    requiredAction               [0] DisplayString
}
```

```
OperatorInformation            ::= SEQUENCE
{
    information                     [0] DisplayString
}
```

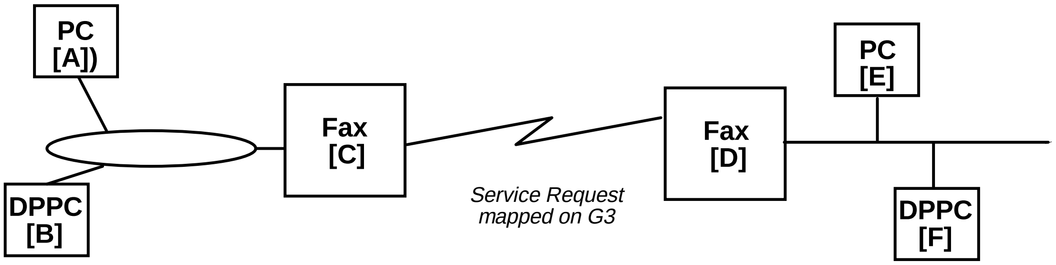
2.5.[Fax Data] Functional Unit

This Functional Unit is newly introduced in V2.0 to enhance a facsimile service in addition to the services provided by [FAX Data Send] Functional Unit. The enhancement focuses on enabling a user to receive a Receipt Notification and Receipt Confirmation when receiving or transmitting a document over T.30 protocol. Here the overview is described. Refer to SLA V2.0 Part-2 Addendum for the complete specification.

2.5.1.1.Overview

[FAX Data Send] Functional Unit of the initial SLA release provides a PC(A) to request a fax transmission to another fax, FAX(D), over standard facsimile protocol such as T.30. A fax at receiving end, FAX(D), may be unaware of a Salutation Architecture, but simply print or receive image data from FAX(C) which is equipped with [FAX Data Send] Functional Unit.

On the other hand, [Fax Data] Functional Unit is introduced into both sending and receiving end, and aware of Salutation Architecture. There are two problems to be solved by introducing [Fax Data] Functional Unit. One is that a client at receiving end has to walk down to a fax equipment to see whether something was faxed to him/her. Second, a client at sender side can not know whether a recipient received the faxed data.



The following describes sample scenarios of the extensions to the current model.

- A client on PC(A) requests [Fax Data] Functional Unit on FAX(C) to fax a document to FAX(D). At the request the client specifies that FAX (D) should notify the receipt completion of the document to PC(E). The notification may be in e-mail or a text message on PC screen. "Receipt Notification" is handled by [Fax Data] Functional Unit of FAX(D).
- When a client on PC(E) accesses the faxed data, "Receipt Confirmation" will be delivered from FAX(D) to FAX(C), and then FAX(C) to PC(A).

In these scenarios data to be transmitted over PSTN could be FAX image data or binary data.

Control information is exchanged between FAX(C) and FAX(D) in addition to data. Necessary protocol will be defined based on standard facsimile protocol.

3.Voice Message Systems

The Voice Message Systems provide a framework not only for storing/exchanging human voice but also for message exchanging between office equipment and persons. In other words, office equipment are regarded as messaging clients in this framework. These messages reach appropriate persons via telephone sets, portable phones or PC speakers.

For example, a fax sends a voice message that it succeeded/failed in sending documents to the person who initiated the job. This allows him/her not to wait in front of the equipment until all the sheets are processed. After putting a stack of sheets, you can ask a copier to notify via telephone when it finishes copying or it had a paper jam.

This kind of systems include three functions:

1. Non-PC/telephone equipment to PC/telephone messaging,
2. PC/telephone to PC/telephone messaging and
3. PC/telephone to non-PC/telephone equipment messaging.

The first two are included in the scope of the Architecture. Voice Message Systems are designed as a framework for the first two functions. Services and attributes of Voice Messaging Systems Functional Unit for the first function will be first defined. The definition will be extended to include the second function in a later release of the architecture.

Voice Message Systems Functional Units will be defined as a common functional model by abstracting broad ranges of Voice/Messaging equipment and by identifying a set of attributes associated with the model. This approach may allow an end user to access to the various size/capability of equipment and to use suitable equipment by capability exchange.

These Functional Units will be coherent with other systems of the Salutation Architecture, namely Document System and Personal Information Systems. This allows voice messages to be processed in the same way as other type of messages, text or images. Namely this allows the Voice Message Systems Functional Units to work as a part of unified mail systems cooperating with Fax mail and electronic mail systems.

PBX may be controlled together with this Voice Message Systems Functional Unit in order to implement above applications. Salutation endorses existing and on-going standards for PBX control. Versit and CSTA of ECMA are included in those. The design will also exploit CMC (Common Messaging Call) definition of folders. It defines services of Functional Units and the attributes of the services.

The following sections describe Voice Message Systems based on the above directions. Portable phones will extend the range of applications of the Voice Message Systems Functional Units. Related Functional Units to them are for further study.

3.1.Voice Message Systems Overview

In today’s office, telephone is one of the most used and important equipment for people to communicate with each other. While you are traveling, it is often the only way. If you cannot get hold of the person you want to talk to, you can leave a voice message at the called telephone in many cases. You can even listen to such messages left for you from another telephone anytime anywhere. With the integration of computer and telephony, other possibilities are rapidly expanding. In fact, what people need to do and are doing to perform day-to-day business with someone else or in a team is the exchange of voice messages, if not documents.

This section addresses services related to voice messages. In this section, the objective of the architecture is to define a standard for applications to provide/use voice message related services to/of another equipment/applications. The goal is to enhance productivity of end users by Salutation Voice Message Systems - the systems that provide voice message related services by implementing the architecture.

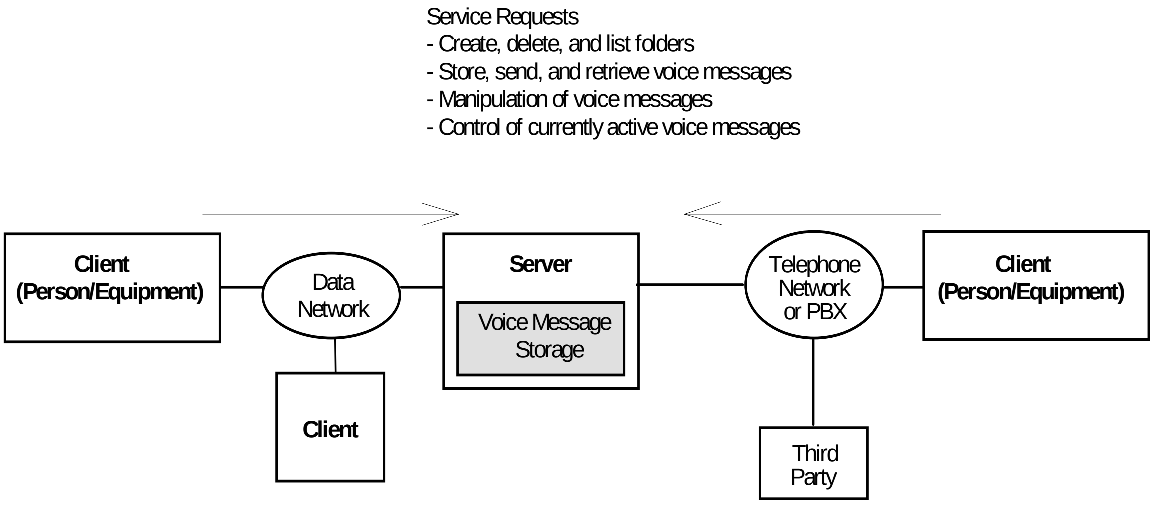
3.1.1.Architecture of Salutation Voice Message Systems

Salutation Voice Message Systems are connected to clients via a telephone network (and/or PBX) and a data network. Voice messages are transferred through either/both of them. A message from a client is generated not only by persons but also by office equipment.

Voice Message Systems should be coherent with the Document Systems in the Salutation Architecture. That is, Voice Message Systems must be manipulated in the same manner as the Document Systems. This enables both the systems to cooperate to construct a unified mail system. Media conversion technology would allow closer unification of Voice Message Systems and Document Systems.

[DOC Storage] Function Unit corresponds to Voice Storage Systems Functional Units. Therefore, the structure of folders in the Voice Message Systems Functional Units could be the same structure as it, of course more powerful and/or complicated structures could be implemented by using commands to the Voice Message Systems Functional Unit.

The following figure illustrates a typical configuration of Salutation Voice Message Systems and outlines.


Service Requests
- Create, delete, and list folders
- Store, send, and retrieve voice messages
- Manipulation of voice messages
- Control of currently active voice messages
Client (Person/Equipment)
Data Network
Client
Server
Voice Message Storage
Telephone Network or PBX
Client (Person/Equipment)
Third Party

- The following Functional Unit is defined in Version 2.0 for the Salutation Voice Message Systems:
  - □ **[Voice Message Storage]**

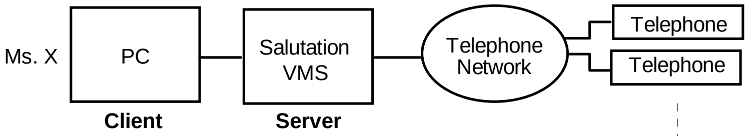
Clients dynamically store/retrieve/delete voice messages to/from this Functional Unit. The storage may be partitioned into several voice message folders.

- A Salutation Voice Message server provides services to clients through either telephone network (e.g. POTS - Plain Old Telephone System -, ISDN, PBX) or data network (e.g. LAN), or both.
- A data network usually provides only data channel. A telephone network always provides voice channel, and may optionally provide data channel. Equipment at both ends must be properly equipped to utilize the data channel in telephone network (e.g. ISDN, voice and data multiplexing technologies).
- The same Salutation Voice Message server may provide a different set of services for voice data transmission over data channel and voice channel to clients.
- Voice messages are transferred between equipment either as "audio" through a voice channel (as you hear a voice message from a telephone answering machine over an ordinary telephone), or as digitized audio data over data channel (which may be played back after digital to analog conversion at the target equipment).
- Some Salutation Voice Message services do not conclude between a Salutation client and a Salutation server, but involve the third-party telephone equipment. An example of such service is that a Salutation client puts a voice message into Salutation server and a list of telephone numbers to request the server to distribute the message to multiple persons.
- Even if a Salutation client and a Salutation server are connected only through a voice channel, a limited Salutation Voice Message Systems services would be still available. The architecture recommends how the Salutation Protocol, which essentially consists of exchanging digital data, should be mapped to the voice channel.

3.1.2.Application Scenarios

Some target application scenarios are shown below to illustrate what services are made possible by the Salutation Voice Message Systems. Each example also shows what and how the Voice Message System Functional Unit commands are used to provide services.

3.1.2.1.Example-1: Voice Message Distribution

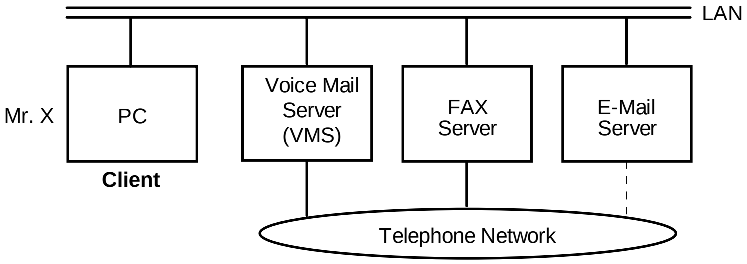


1) The director in charge of R&D on the west coast is suddenly called by the headquarters on the east coast. The budget planning meeting to be held tomorrow will be postponed.
2) His secretary, Ms. X, creates a voice message using her PC to tell the meeting of tomorrow will be postponed, selects the project leaders from the electric phone directory, and requests the Salutation Voice Message Systems to distribute the message.
3) She always used to call all project leaders, 23 at present, in such occasions. Now, she only needs to create a message once and picks up a distribution list. If necessary, she can later collect who is attending the meeting and who is not, using the push buttons on the telephone.

The following services of the Functional Unit will be used:

- Store voice message
- Send voice message

3.1.2.2.Example-2: Integrated Mail Box for Voice Mail, E-Mail, and FAX



1) Mr. X turns on his PC every morning. The Salutation mail box application starts automatically to check all mails addressed to him and presents the list.

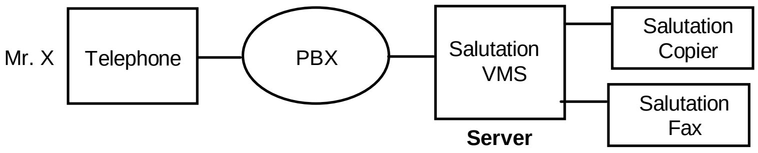
2) This morning, he has received two voice mails, eight e-mails, and one FAX mail, though he does not really care which is which these days. It seems one of the voice mails is from an external Non-Salutation telephone as it does not show the sender's name, subject, etc. in the list.

3) He retrieves the voice mail message through the speaker on his PC, and finds that it is a complaint from one of his customers. He forwards it with his comment to the responsible department, with "Urgent" mark on.

The following services of the Functional Unit will be used:

- List Folder Content (to get a list of received Voice Messages)
- Retrieve Voice Message (to retrieve a selected Voice Message)
- Create Voice Message (to create his comments)
- Concatenate Voice Message (to combine his comments with the customer's Voice Message)
- Send Voice Message (to forward the Voice Message)

3.1.2.3.Example-3: Equipment Status Inquiry/Report



1) Mr. X needs to FAX a very thick document.

2) He walks to the Salutation FAX machine, sets the original document, pushes appropriate buttons including his telephone number and starts the machine.

3) Several minutes after he returns to his desk, his telephone rings. It is from the FAX machine to tell him in voice that a paper jam has occurred.
4) He goes back to the FAX machine, fixes the problem, and restarts the machine.
5) His telephone rings again after a while. This time he is notified of the successful completion of the job.

The following service of the Functional Unit will be used:

- Play Voice Message

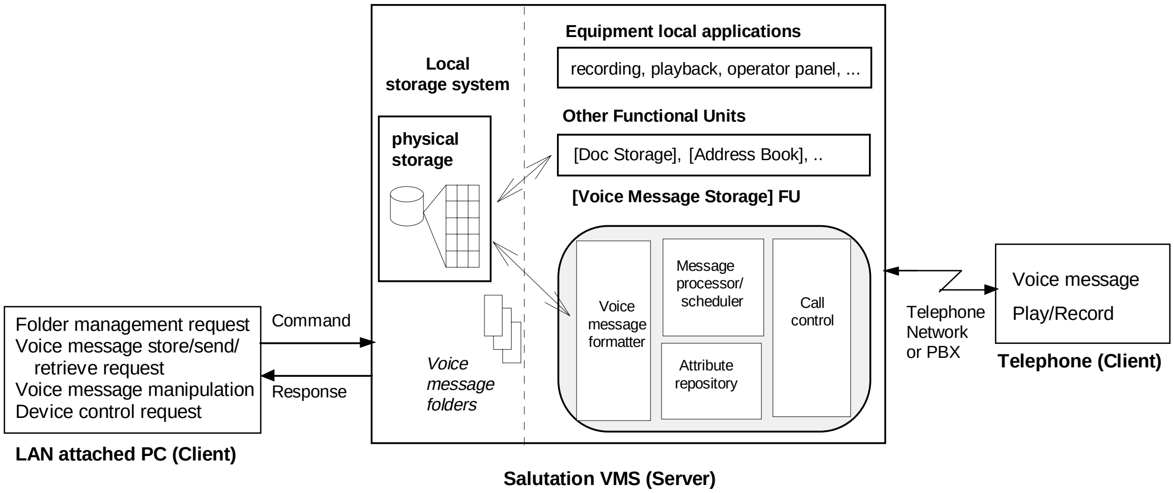
The same example also applies to the scenario in which a user requests a copy machine to generate copies of a thick document, and the copy machine notifies failure/success of the user's request by making a telephone call to the user.

3.2.[Voice Message Storage] Functional Unit

3.2.1.Overview

[Voice Message Storage] Functional Unit provides the interfaces for a client to handle voice messages. It defines voice message handling services like those for storing voice message, deleting voice message, sending voice message , and/or retrieving voice message, which are typically found in modern telephone answering machines and voice mail systems. It also abstracts the storage device as a container of voice messages that may be partitioned into multiple folders.

The following figure illustrates a configuration model to understand how [Voice Message Storage] Functional Unit works with other resources within equipment and with remote clients who issue service requests.


Equipment local applications
recording, playback, operator panel, ...
Local storage system
Other Functional Units
[Doc Storage], [Address Book], ..
physical storage
[Voice Message Storage] FU
Message processor/ scheduler
Voice message formatter
Call control
Attribute repository
Voice message folders
Folder management request
Voice message store/send/ retrieve request
Voice message manipulation
Device control request
Command
Response
LAN attached PC (Client)
Telephone Network or PBX
Voice message Play/Record
Telephone (Client)
Salutation VMS (Server)

[Voice Message Storage] Functional Unit is considered to be composed of the following logical sub-components or service elements.

- Front-end message processor and job scheduler

- Attribute Repository manager
- Voice message formatter
- Virtual storage manager (directory manager)
- Call control

3.2.2.Two phase design of [Voice Message Storage] Functional Unit

The design of [Voice Message Storage] Functional Unit takes phasing approach:

1. The first phase defines only those services and attributes that are necessary for non-PC/telephone equipment to PC/telephone messaging, and
2. The second phase will define additional services and attributes that are needed for PC/telephone to PC/telephone messaging.

The services and attributes defined in the first phase form a subset of the fullset services and capabilities to be defined in the second phase. In the following description, it is referred to the first phase part as **Subset [Voice Message Storage] FU** and the fullset as **Fullset [Voice Message Storage] FU**. This release of the specification defines the Subset [Voice Message Storage] FU in detail and provides some idea of the Fullset [Voice Message Storage] FU to allow the readers to visualize the power of Salutation [Voice Message Storage] FU.

3.2.3.Subset [Voice Message Storage] FU

3.2.3.1.List of Functional Unit Attributes for Subset [Voice Message Storage] FU

The following capability attributes are defined in the Subset [Voice Message Storage] FU:

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private/ Job Attribute
personalityProtocol	20000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No/No
supportLevel	20001	N/A	INTEGER -value should be 'one' for Subset of [Voice Message Storage] FU (intGreaterThanOrEqualTo)	Yes	No/No
supportedCommand	20002	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No/No
dynamicStatusId	20003	N/A	SET OF DynamicStatusID (setIntDoesContain)	No	No/No
maxDuration	20020	INTEGER	INTEGER - max value (intGreaterThanOrEqualTo)	Yes	No/No
maxReceiversPlay	20021	Receiver	INTEGER - max number of receivers (intGreaterThanOrEqualTo)	Yes	No/No
maxRecipientsSend	20022	Recipient	INTEGER - max number of recipients (intGreaterThanOrEqualTo)	Yes	No/No

voiceSpeed	20023	INTEGER	BOOLEAN (boolEqualTo)	No	No/No
voiceVolume	20024	INTEGER	BOOLEAN (boolEqualTo)	No	No/No
deliveryGrade	20025	DeliveryGrade	SET OF DeliveryGrade (setIntDoesContain)	Yes	No/No
priorityLevel	20030	PriorityLevel	SET OF PriorityLevel (setIntDoesContain)	Yes	No/Yes
copyRecipients	20040	Recipient	BOOLEAN (boolEqualTo)	No	No/No
blindCopyRecipients	20041	Recipient	BOOLEAN (boolEqualTo)	No	No/No
deferredDeliveryTime	20042	UTCTime	BOOLEAN (boolEqualTo)	No	No/No
subject	20043	DisplayString	BOOLEAN (boolEqualTo)	No	No/No
maxSubjectLength	20044	N/A	INTEGER- max length of subject (intGreaterThanOrEqualTo)	Yes	No/No
synthesize	20050	N/A	BOOLEAN (boolEqualTo)	No	No/No
synthesizeVoiceSpeed	20051	INTEGER	BOOLEAN (boolEqualTo)	No	No/No
synthesizeVoiceVolume	20052	INTEGER	BOOLEAN (boolEqualTo)	No	No/No
synthesizeVoiceType	20053	INTEGER	SET OF VoiceType (setIntDoesContain)	Yes	No/No
synthesizeTextLanguage	20054	TextLanguage	SET OF TextLanguage (setIntDoesContain)	Yes	No/No
encoding	20060	Encoding	SET OF Encoding (setIntDoesContain)	Yes	No/No
minimumCheckInterval -- the minimum allowed -- value to be set in the -- checkInterval parameter -- of a SubscribeEvent -- command	20070	N/A	INTEGER (intGreaterThanOrEqualTo)	No	No/No

NOTE: [Voice Message Storage] FU defines a standard range (0 to 10, with 0 being the lowest and 10 being the highest) for voiceSpeed, voiceVolume, synthesizeVoiceSpeed and synthesizeVoiceVolume. A user can specify any value in this range for the parameters corresponding to these attributes in [Voice Message Storage] FU commands.

3.2.3.2.Salutation Personality Message & Protocol for Subset [Voice Message Storage] FU

This section describes service request protocol for Subset [Voice Message Storage] FU under Salutation Personality.

3.2.3.2.1.Request Procedure for Subset [Voice Message Storage] FU

3.2.3.2.1.1.Commands of Subset [Voice Message Storage] FU

The following commands and responses are used in the Subset [Voice Message Storage] FU for making job requests.

- [Voice Message Storage] FU Mandatory support Command

  For Folder management services

  □ ListFolderContentVM

  For Data I/O services

  □ SendVM

  □ PlayVM

- [Voice Message Storage] FU Mandatory support common Commands

  The following common commands and responses should be supported.

  □ RequestDataTransfer

  □ DataBlockDescription

  □ TransferDataBlock

  □ RequestNextData

  □ ACK

  □ NACK

- [Voice Message Storage] FU Optional support Command

  For Message Manipulation services

  □ SynthesizeVM

  For Vendor specific services

  □ VendorEscape

3.2.3.2.2.Subset [Voice Message Storage] FU command details

3.2.3.2.2.1.ListFolderContentVM

Client	**Server**
ListFolderContentVM =>	
	<= TransferDataBlock (VoiceMsgList)
ACK (NULL) =>	

This command is used to get a list of the voice messages stored in a specified folder of the Voice Message Systems device.

A user may use this command to first list out the contents of a folder to get an idea of the voice messages that it contains, then select a particular voice message from the list, and play it or send it to another user.

```
ListFolderContentVM          ::= [APPLICATION tagListFolderContentVM] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    folderId                     [0] FolderID
}

FolderID                     ::= INTEGER
                                 -- FolderID=0 is used for Default Public Folder.
```

Data transferred by TransferDataBlock command is as follows:

```
VoiceMsgList                 ::= SET OF VoiceMessageDescriptor

VoiceMessageDescriptor       ::= SEQUENCE
{
    voiceMsgId                   [0] VoiceMsgID,
    descriptiveComment           [1] DescriptiveComment    OPTIONAL
}

DescriptiveComment           ::= DisplayString
```

NACK Response

Name	**Description**	**ReturnCode**
rcFolderNotFound	Specified folder does not exist	138
rcFolderAccessRejected	Access to folder has not been authorized	139

Sample protocol sequences are provided below.

Example Protocol Sequence (1)

Client	**Server**

A user uses ListFolderContentVM and then PlayVM with explicit VoiceMsgID.

ListFolderContentVM(...)=>

<= TransferDataBlock (VoiceMsgList)

ACK (NULL) =>

A user selects a voice message from the list.

PlayVM(...)=>

Example Protocol Sequence (2)

Client	**Server**

A user uses ListFolderContentVM and then SendVM with explicit VoiceMsgID.

ListFolderContentVM(...)=>

<= TransferDataBlock (VoiceMsgList)

ACK (NULL) =>

A user selects a voice message from the list.

SendVM(...)=>

Example Protocol Sequence (3)

Client	**Server**

A sender uses ListFolderContentVM and then SendVM with explicit VoiceMsgID. The receiver uses ListFolderContentVM and then PlayVM to select and listen to the received voice mail.

A sender uses ListFolderContentVM.

ListFolderContentVM(...)=>

<= TransferDataBlock (VoiceMsgList)

ACK (NULL) =>

A sender selects a voice message from the list.

SendVM(...)=>

A receiver uses ListFolderContentVM to get a list of received voice messages.

ListFolderContentVM(...)=>

<= TransferDataBlock (VoiceMsgList)

ACK (NULL) =>

A receiver selects a voice message from the list.

PlayVM(..., OwnTelNo, ...)=>

3.2.3.2.2.2.SendVM

Client	Server

SendVM =>

<= ACK(JobHandle)/NACK(ReturnCode)

This command is used to send a specified voice message to one/more receivers. It is like sending a voice mail to a receiver. The sent voice message gets stored in the appropriate folder of the receiver in a Voice Message Systems device.

A user may use this command to send an already existing voice message to another user. For example, a user may use this command to forward a received voice message to all concerned users. A user may also use this command to send a newly created voice message to one/more users.

```
SendVM                          ::= [APPLICATION tagSendVM] SEQUENCE
{                                   COMPONENTS OF MsgHeader,
    folderId                        [0] FolderID,
    voiceMsgId                      [1] VoiceMsgID,
    recipients                      [2] SET OF Recipient,
    deliveryGrade                   [3] DeliveryGrade           OPTIONAL,
                                        -- sender specifies the grade of delivery. This information
                                        -- is for the mail server
    priorityLevel                   [4] SimpleJobPriority       OPTIONAL,
                                        -- sender specifies the priority level of the message. This
                                        -- information is for the receiver
    subject                         [5] DisplayStringOPTIONAL
                                        -- sender specifies the subject of the message.
                                        -- maximum 256 characters
                                        -- parameters below this are not needed in the subset,
                                        -- but may be needed in the fullset. Of course, which
                                        -- of these are actually needed is an item for further
                                        -- study
    -- alternateRecipientAllowed    [11] BOOLEAN                OPTIONAL,
                                        -- are other users allowed to receive the message in addition
                                        -- to the recipient
    -- authorizingUsers             [12] SET OF UserID          OPTIONAL,
                                        -- users who authorized the sender to send the message
                                        -- (upto maximum 16 users)
    -- conversionWithLossProhibited [13] BOOLEAN                OPTIONAL,
                                        -- sender instructs that implicit encoded information
                                        -- type conversion(s) should not be performed if there is
                                        -- any possibility of information loss
    -- crossReferences                 [14] SET OF VoiceMsgID  OPTIONAL,
                                           -- cross refer to the specified voice messages
                                           -- (upto maximum 8 VoiceMsgIDs)
    -- expiryDate                      [15] UTCTime            OPTIONAL,
                                           -- sender instructs the expiry date and time of the message
    -- implicitConversionProhibited        [16] BOOLEAN                OPTIONAL,
                                           -- sender instructs that implicit encoded information
                                           -- type conversion(s) should not be performed
    -- inReplyTo                       [17] VoiceMsgID         OPTIONAL,
                                           -- sender specifies in-reply-to VoiceMsgID
    -- latestDeliveryTime              [18] UTCTime            OPTIONAL,
                                           -- date and time by which the message must be delivered
                                           -- to the receiver(s)
    -- nonReceiptNotificationRequest   [19] BOOLEAN            OPTIONAL,
                                           -- sender specifies whether he/she wants to be notified or not
                                           -- if the message is not received by the receiver(s)
    -- obsoletes                       [20] SET OF VoiceMsgID  OPTIONAL,
                                           -- sender specifies obsolete messages (maximum 8)
    -- preventionOfNonDeliveryNotification  [21] BOOLEAN       OPTIONAL,
                                           -- do not return a non-delivery notification to the sender
                                           -- if the message cannot be delivered
    -- receiptNotificationRequest      [22] BOOLEAN            OPTIONAL,
                                           -- sender wants to be notified when the message has been
                                           -- received by the recipient(s)
    -- redirectionDisallowed           [23] BOOLEAN            OPTIONAL,
                                           -- sender specifies that the redirection of the message
                                           -- should not be done if the recipient has requested this
    -- replyRequest                    [24] BOOLEAN            OPTIONAL,
                                           -- sender requests for a reply from the recipient(s)
    -- replyBy                         [25] UTCTime            OPTIONAL,
                                           -- sender specifies the deadline for replying
    -- replyToUsers                    [26] SET OF UserID      OPTIONAL,
                                           -- sender specifies the user(s) whom to send reply
                                           -- (maximum 32 users)
    -- sensitivity                     [27] Sensitivity        OPTIONAL
                                           -- sender specifies the sensitivity level of the message
}

VoiceMsgID                         ::= INTEGER
```

```
Recipient                     ::= SEQUENCE
{
    jobEntryId                    [0] JobEntryID,
    recipientId                   [1] UserID,
    recipientType                 [2] ENUMERATED
    {
        primary                   (0),
        copy                      (1),
        blindCopy                 (2)
    },
    deferredDeliveryTime          [3] UTCTime              OPTIONAL
                                      -- sender specifies the date and time up to which delivery
                                      -- of the message should be deferred
}

DeliveryGrade                 ::= ENUMERATED
{
    urgent                        (0),
    normal                        (1),
    nonUrgent                     (2)
}
```

ACK Response

JobHandle

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	138
rcFolderAccessRejected	Access to folder has not been authorized	139
rcInvalidVoiceMsgId	Specified Voice Message not found	148
rcInvalidRecipientId	Specified recipientId is invalid	168
rcInvalidRecipientType	Specified recipientType is invalid	169
rcInvalidDeferredDeliveryTime	Specified deferredDeliveryTime is invalid	188
rcInvalidDeliveryGrade	Specified delivery grade not valid	189
rcInvalidPriorityLevel	Specified priority level not valid	190
rcInvalidSubject	Specified subject not valid	191

Sample protocol sequences are provided below.

Example Protocol Sequence (1)

Client	Server

A user uses ListFolderContentVM and then SendVM with explicit VoiceMsgID.

ListFolderContentVM(...)=>

<= TransferDataBlock(VoiceMsgList)

ACK (NULL) =>

A user selects a voice message from the list.

SendVM(...)=>

Example Protocol Sequence (2)

Client	**Server**

A sender uses ListFolderContentVM and then SendVM with explicit VoiceMsgID. The receiver uses ListFolderContentVM and then PlayVM to select and listen to the received voice mail.

A sender uses ListFolderContentVM.

ListFolderContentVM(...)=>

<= TransferDataBlock(VoiceMsgList)

ACK (NULL) =>

A sender selects a voice message from the list.

SendVM(...)=>

A receiver uses ListFolderContentVM to get a list of received voice messages.

ListFolderContentVM(...)=>

<= TransferDataBlock(VoiceMsgList)

ACK (NULL) =>

A receiver selects a voice message from the list.

PlayVM(..., OwnTelNo, ...)=>

Example Protocol Sequence (3)

Client	**Server**

A user uses SynthesizeVM and then SendVM with the VoiceMsgID of the newly synthesized voice message

SynthesizeVM(...)=>

<= ACK(VoiceMsgID)

SendVM(...)=>

3.2.3.2.2.3.PlayVM

Client	**Server**

PlayVM =>

<= ACK(JobHandle)/NACK(ReturnCode)

This command is used to start playing a voice message on a telephone connection line.

A user may use this command either to listen to an already existing voice message using his/her own telephone, or to send a voice message to another user using voice channel.

```
PlayVM                      ::= [APPLICATION tagPlayVM] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    folderId                    [0] FolderID,
    voiceMsgId                  [1] VoiceMsgID,
    receivers                   [2] SET OF Receiver,
    headerInformation           [3] HeaderInformation     OPTIONAL,
    voiceDuration               [4] INTEGER               OPTIONAL,
    voiceSpeed                  [5] INTEGER               OPTIONAL,
    voiceVolume                 [6] INTEGER               OPTIONAL
}

Receiver                    ::= SEQUENCE
{
    jobEntryId                  [0] JobEntryID,
    receiverId                  [1] CHOICE
    {
        userId                  [0] UserID,
        telephoneNo             [1] TelephoneNumberString
    },
    deferredDeliveryTime        [2] UTCTime               OPTIONAL
                                    -- sender specifies the date and time up to which delivery
                                    -- of the message should be deferred for this receiver
}

HeaderInformation           ::= BIT STRING
{
    senderId                    (0),
    dateSent                    (1)
}
```

ACK Response

JobHandle --- is needed to perform other operations on the voice message during PlayVM.

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	138
rcFolderAccessRejected	Access to folder has not been authorized	139
rcInvalidVoiceMsgId	Specified Voice Message not found	149
rcInvalidReceiverId	Specified receiverId is not valid	170
rcInvalidDeferredDeliveryTime	Specified deferredDeliveryTime is not valid	188
rcInvalidHeaderInfo	Specified header information not valid	192
rcInvalidVoiceDuration	Specified voice duration not valid	200
rcInvalidVoiceSpeed	Specified voice speed not valid	201
rcInvalidVoiceVolume	Specified voice volume not valid	202

Sample protocol sequences are provided below.

Example Protocol Sequence (1)

Client	Server

A user uses ListFolderContentVM and then PlayVM with explicit VoiceMsgID and OwnTelNo to listen to the voice message.

ListFolderContentVM(...)=>

<= TransferDataBlock(VoiceMsgList)

ACK (NULL) =>

A user selects a voice message from the list.

PlayVM(..., OwnTelNo, ...)=>

Example Protocol Sequence (2)

Client	Server

A user uses ListFolderContentVM and then PlayVM with explicit VoiceMsgID and AnotherUserTelNo to send the voice message to that user via voice channel.

ListFolderContentVM(...)=>

<= TransferDataBlock(VoiceMsgList)

ACK (NULL) =>

A user selects a voice message from the list.

PlayVM(..., AnotherUserTelNo, ...)=>

Example Protocol Sequence (3)

Client	Server

A sender uses ListFolderContentVM and then SendVM with explicit VoiceMsgID. The receiver uses ListFolderContentVM and then PlayVM to select and listen to the received voice mail.

A sender uses ListFolderContentVM.

ListFolderContentVM(...)=>

<= TransferDataBlock(VoiceMsgList)

ACK (NULL) =>

A sender selects a voice message from the list.

SendVM(...)=>

A receiver uses ListFolderContentVM to get a list of received voice messages.

ListFolderContentVM(...)=>

<= TransferDataBlock(VoiceMsgList)

ACK (NULL) =>

A receiver selects a voice message from the list.

PlayVM(..., OwnTelNo, ...)=>

Example Protocol Sequence (4)

Client	Server

A user uses SynthesizeVM and then PlayVM with the VoiceMsgID of the newly synthesized voice message.

SynthesizeVM(...)=>

<= ACK(VoiceMsgID)

PlayVM(...)=>

3.2.3.2.2.4.SynthesizeVM

Client	Server

SynthesizeVM =>

<= ACK(VoiceMsgID)/NACK(ReturnCode)

This command is used to construct a voice message from a text message and store it in the specified folder. When operation is complete, VoiceMsgID assigned to the voice message is returned.

A user may use this command when he/she does not has any voice recording facility in the equipment that he/she is using, but wants to create/send a voice message. He/she can do this by first writing the message in text form, and then using the SynthesizeVM command to get the voice message equivalent of the text message. This command is also useful for sending voice messages from an equipment to a person. This is because an equipment may have limited storage that can store messages only in text form and not voice form. When sending a message to a user, the equipment may send the text form of the message with a request to deliver it in voice form. In this case, the system automatically converts the text message into a voice message by using the SynthesizeVM command.

```
SynthesizeVM                    ::= [APPLICATION tagSynthesizeVM] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    folderId                        [0] FolderID,
    text                            [1] DisplayString,
    textLanguage                    [2] TextLanguage            OPTIONAL,
    voiceMessageDataDescriptor      [3] VoiceMessageDataDescriptor      OPTIONAL,
    voiceType                       [4] VoiceType               OPTIONAL,
    voiceSpeed                      [5] INTEGER                 OPTIONAL,
    voiceVolume                     [6] INTEGER                 OPTIONAL
}

TextLanguage                    ::= DisplayString
                                    -- Language tag which is defined in RFC 1766.
                                    -- Language tag consists of primary tag which is ISO 639
                                    -- language and secondary tag which is ISO 3166 country/area
                                    -- in which the language is used.

VoiceMessageDataDescriptor      ::= SEQUENCE
{
    voiceMessageDataFormat          [0] VoiceMessageDataFormat,
    voiceMessageFormatInterpretation    [1] VoiceMessageFormatInterpretation
}

VoiceMessageDataFormat          ::= ENUMERATED
{
    voiceMessage                    (0)
}

VoiceMessageFormatInterpretation    ::= CHOICE
{
    voiceMessageEncoding            [0] Encoding
}
```

```
Encoding                    ::= SEQUENCE
{
    encodingAlgorithm           [0] EncodingAlgorithm,
    samplingRate                [1] SamplingRate          OPTIONAL
}

EncodingAlgorithm           ::= ENUMERATED
{
    analog                      (0),
    pcm                         (1),
    u-law                       (2),
    a-law                       (3),
    adpcm                       (4),
    cvsd                        (5),
    apc-ab                      (6),
    ld-celp                     (7),
    v-celp                      (8),
    others                      (127)
}

SamplingRate                ::= ENUMERATED
{
--  r4K                         (0),
--  r8K                         (1),
--  r16K                        (2),
    r24K                        (3),
    r32K                        (4)
-- r64K                         (5),
-- others                       (127)
}

VoiceType                   ::= ENUMERATED
{
    maleVoicePreferred          (125),
    femaleVoicePreferred        (126),
    dontCare                    (127)
}
```

ACK Response

VoiceMsgID of the new voice message

NACK Response

Name	**Description**	**ReturnCode**
rcFolderNotFound	Specified folder does not exist	138
rcFolderAccessRejected	Access to folder is not authorized	139
rcInvalidText	Text data is not valid	218

rcInvalidTextLanguage	Text language is not valid	210
rcInvalidEncodingAlgo	Specified encoding algorithm not valid	203
rcInvalidSamplingRate	Specified sampling rate not valid	204
rcInvalidVoiceType	Specified voice type not valid	205
rcInvalidVoiceMessageDescriptor	Specified VoiceMessageDescriptor not valid	206
rcInvalidVoiceMessageDataFormat	Specified VoiceMessageDataFormat not valid	207
rcInvalidVoiceMessageFormatInterpretation	Specified VoiceMessageFormatInterpretation not valid	208
rcInvalidVoiceSpeed	Specified voice speed not valid	201
rcInvalidVoiceVolume	Specified voice volume not valid	202

Sample protocol sequences are provided below.

Example Protocol Sequence (1)

Client	**Server**

A user uses SynthesizeVM and then PlayVM with the VoiceMsgID of the newly synthesized voice message.

SynthesizeVM(...)=>

<= **ACK(VoiceMsgID)**

PlayVM(...)=>

Example Protocol Sequence (2)

Client	**Server**

A user uses SynthesizeVM and then SendVM with the VoiceMsgID of the newly synthesized voice message.

SynthesizeVM(...)=>

<= **ACK(VoiceMsgID)**

SendVM(...)=>

Example Protocol Sequence (3)

Client	**Server**

A sender uses SynthesizeVM, then PlayVM and then SendVM with the VoiceMsgID of the newly synthesized voice message. The receiver then uses ListFolderContentVM and then PlayVM to select and listen to the received voice mail.

A sender uses SynthesizeVM.

SynthesizeVM(...)=>

<= ACK(VoiceMsgID)

A sender uses PlayVM to confirm if the synthesized voice message is OK.

PlayVM(..., OwnTelNo, ...)=>

A sender then sends the voice message to the receiver.

SendVM(...)=>

A receiver next uses ListFolderContentVM to get a list of received voice messages.

ListFolderContentVM(...)=>

<= TransferDataBlock(VoiceMsgList)

ACK (NULL) =>

A receiver selects the voice message from the list.

PlayVM(..., OwnTelNo, ...)=>

3.2.3.2.3.Dynamic Status Operations

Dynamic Status operations allow a client to know the aspect of Functional Unit and the transition in the aspects. **Dynamic Status Parameter** describes the aspects. A client may query the current values of Dynamic Status Parameter, and request [Voice Message Storage] FU to notify an Event when any transition occurs in the values of Dynamic Status Parameter.

The following commands and response are used for dynamic status operations. The usage of those commands and responses are described in "**Dynamic Status Messages**" section on page 49.

- [Voice Message Storage] FU Mandatory support common command
  - □ QueryDynamicStatus
  - □ ACK and NACK
- [Voice Message Storage] FU Optional support common command
  - □ SubscribeEvent, UnsubscribeEvent and NotifyEvent (These commands belong to the same Optional group, so an FU must support all these commands if it supports them.)

The following Dynamic Status Parameters are defined for [Voice Message Storage] Functional Unit.

Dynamic Status Parameter	Query	Event	ID	Description
PlayVMStatus	Yes	Yes	20000	Status of play voice message

Data Type of Dynamic Status Parameter

```
PlayVMStatus                    ::= ENUMERATED
{
    playing                         (0),
    suspended                       (1),
    position                        (2),
    error                           (3),
    others                          (127)
}
```

3.2.3.2.4.Job Related Operations

A client application can control the way of executing a job and also know the status of the job execution. The usage of those commands and responses are described in "**Job-Related Messages**" section on page 33.

3.2.3.2.4.1.Controlling Job execution

[Voice Message Storage] Functional Unit defines priorityLevel attribute as **Job Control Attributes**. The following commands may be used to change the value of the attributes or cancel a job or job entry.

- [Voice Message Storage] FU Mandatory support Command
    - □ CancelJob
    - □ FreeJobHandle
    - □ ChangeJobAttribute
    - □ ACK and NACK
- [Voice Message Storage] FU Optional support Command
    - □ CancelJobEntry
    - □ ChangeJobEntryAttribute

Note) CancelJobEntry, ChangeJobEntryAttribute, QueryJobEntryStatus, NotifyJobEntryStatus, SuspendJobEntry, and ResumeJobEntry belong to the same Optional command Group, so an FU must support all these commands if it supports them.

3.2.3.2.4.2.Job Status Notification

[Voice Message Storage] Functional Unit provides flexible ways for a client to know the status or the result of Voice Message Operation request.

The following commands and responses are used for job status notification.

- [Voice Message Storage] FU Mandatory support Command
    - □ QueryJobStatus
    - □ ACK and NACK
- [Voice Message Storage] FU Optional support Command
    - □ NotifyJobStatus

Note) NotifyJobStatus, StartMonitorJobStatus and CancelMonitorJobStatus belong to the same Optional command Group, so an FU must support all these commands if it supports them.

- □ QueryJobEntryStatus
- □ NotifyJobEntryStatus

Note) CancelJobEntry, ChangeJobEntryAttribute, QueryJobEntryStatus, NotifyJobEntryStatus, SuspendJobEntry, and ResumeJobEntry belong to the same Optional command Group, so an FU must support all these commands if it supports them.

3.2.3.2.4.3.Job Entry Suspend/Resume

[Voice Message Storage] Functional Unit supports the following commands to suspend/resume jobs submitted by “PlayVM” command (subset) or RecordVM command (fullset).

- ● [Voice Message Storage] FU Mandatory support Command
  - □ SuspendJob
  - □ ResumeJob
- ● [Voice Message Storage] FU Optional support Command
  - □ SuspendJobEntry
  - □ ResumeJobEntry

Note) CancelJobEntry, ChangeJobEntryAttribute, QueryJobEntryStatus, NotifyJobEntryStatus, SuspendJobEntry, and ResumeJobEntry belong to the same Optional command Group, so an FU must support all these commands if it supports them.

3.2.3.2.4.4.Job Status Monitor Start/Cancel

[Voice Message Storage] Functional Unit supports the following commands to start/cancel job-status-monitoring.

- ● [Voice Message Storage] FU Optional support common command
  - □ StartMonitorJobStatus
  - □ CancelMonitorJobStatus

Note) NotifyJobStatus, StartMonitorJobStatus and CancelMonitorJobStatus belong to the same Optional command Group, so an FU must support all these commands if it supports them.

3.2.3.2.4.5.List FU Job Status

- ● [Voice Message Storage] FU Mandatory support Command
  - □ ListVMSJob
  - □ ACK and NACK

ListVMSJob Command

ListVMSJob command is used to get the list of job in the **[Voice Message Storage]** Functional Unit.

ASN.1 Syntax Definition

```
ListVMSJob                ::= [APPLICATION tagListVMSJob] SEQUENCE
{
                              COMPONENTS OF MsgHeader
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcNoJob	There is no job	128

Data transferred by TransferDataBlock command is as follows;

```
VMSJobList                  ::= SET OF VMSJobDescription

VMSJobDescription         ::= SEQUENCE
{
    jobHandle                   [0] JobHandle,
    jobStatusCode               [1] JobStatusCode,
    numOfJobEntries             [2] INTEGER
}
```

3.2.3.2.4.6.Job-Specific Reason code

The [Voice Message Storage] Functional Unit specific reason codes will be returned in a NotifyJobStatus, NotifyJobEntryStatus, and ACK response to QueryJobStatus or QueryJobEntryStatus Command.

Name	Description	ReasonCode
equipmentError	terminated due to equipment detected errors.	128
waitingForRetry	in waiting mode for retry call.	129

3.2.4.Fullset [Voice Message Storage] FU

As mentioned before, the Fullset [Voice Message Storage] FU is an expanded version of the Subset [Voice Message Storage] FU that defines commands and attributes for PC/telephone to PC/telephone messaging. Therefore, all the commands and attributes defined in the Subset [Voice Message Storage] FU are also a part of the Fullset [Voice Message Storage] FU. However, the commands and attributes already presented in the description of Subset [Voice Message Storage] FU will not be repeated again and only the additional ones are presented below.

Note: *The commands and attributes described below are not yet finalized and are being further studied and refined. They are given here simply to give an idea of the power of the full specification of Salutation [Voice Message Storage] FU.*

3.2.4.1.List of Functional Unit Attributes for Fullset [Voice Message Storage] FU

The following additional capability attributes are tentatively defined in the Fullset [Voice Message Storage] FU:

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private/ Job Attribute
folderType		FolderType	SET OF FolderType (setIntDoesContain)	Yes	No/No
maxDescriptiveComment		N/A	INTEGER - max value (IntGreaterThanOrEqualTo)	Yes	No/No
accessMode		AccessMode	SET OF AccessMode (setIntDoesContain)	Yes	No/No
voiceLevel		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
notifyType		NotifyType	SET OF NotifyType (setIntDoesContain)	Yes	No/No
sensitivity		Sensitivity	SET OF Sensitivity (setIntDoesContain)	Yes	No/No
alternateRecipientAllowed		INEGER	BOOLEAN (boolEqualTo)	No	No/No
authorizingUsers		UserID	BOOLEAN (boolEqualTo)	No	No/No
maxAuthorizingUsers		N/A	INTEGER - max value (IntGreaterThanOrEqualTo)	Yes	No/No
conversionWithLossProhibited		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
crossReferences		VoiceMsgID	BOOLEAN (boolEqualTo)	No	No/No
maxCrossReferences		N/A	INTEGER - max value (IntGreaterThanOrEqualTo)	Yes	No/No
expiryDate		UTCTime	BOOLEAN (boolEqualTo)	No	No/No
implicitConversionProhibited		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
inReplyTo		VoiceMsgID	BOOLEAN (boolEqualTo)	No	No/No
latestDeliveryTime		UTCTime	BOOLEAN (boolEqualTo)	No	No/No
nonReceiptNotificationRequest		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
obsoletes		VoiceMsgID	BOOLEAN (boolEqualTo)	No	No/No
maxObsoletes		N/A	INTEGER - max value (IntGreaterThanOrEqualTo)	Yes	No/No

preventionOfNonDelivery Notification		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
ReceiptNotificationRequest		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
redirectionDisallowed		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
replyRequest		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
replyBy		UTCTime	BOOLEAN (boolEqualTo)	No	No/No
replyToUsers		UserID	BOOLEAN (boolEqualTo)	No	No/No
maxReplyToUsers		N/A	INTEGER - max value (IntGreaterThanOrEqualTo)	Yes	No/No
autoForwarded[4]		AutoForwarded	BOOLEAN (boolEqualTo)	No	No/No
dlExpansionHistory4		DlExpansionHistory	BOOLEAN (boolEqualTo)	No	No/No
holdForDelivery4		HoldForCriteria	BOOLEAN (boolEqualTo)	No	No/No
implicitConversion4		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
redirectionAddress4		Recipient	BOOLEAN (boolEqualTo)	No	No/No
restrictedDeliveryId4		Recipient	BOOLEAN (boolEqualTo)	No	No/No
storedMessageAlert4		INTEGER	BOOLEAN (boolEqualTo)	No	No/No
autoForwardAddress4		Recipient	BOOLEAN (boolEqualTo)	No	No/No
submissionTimestamp4		UTCTime	BOOLEAN (boolEqualTo)	No	No/No
autoSubmitted4		INTEGER	BOOLEAN (boolEqualTo)	No	No/No

3.2.4.2.Salutation Personality Message & Protocol for Fullset [Voice Message Storage] FU

This section describes service request protocol for Fullset [Voice Message Storage] FU under Salutation Personality.

[4] These parameters are settable for message receivers.

3.2.4.2.1.Command Request Procedure for Fullset [Voice Message Storage] FU

3.2.4.2.1.1.Commands of Fullset [Voice Message Storage] FU

The following list describes the tentative additional commands for the Fullset [Voice Message Storage] FU.

Folder management services

- □ CreateFolderVM
- □ DeleteFolderVM
- □ ChangeFolderDescVM

Data I/O services

- □ StoreVM
- □ RecordVM
- □ RetrieveVM
- □ SetReceiverOptionsVM

Message Manipulation services

- □ DeleteVM
- □ CopyVM
- □ ConcatenateVM
- □ SeparateVM

Device Control services

- □ RepositionVM
- □ ReviewVM

3.2.4.2.1.2.Common commands

The following common commands and responses are also used.

- □ RequestDataTransfer
- □ DataBlockDescription
- □ TransferDataBlock
- □ RequestNextData
- □ VendorEscape
- □ ACK
- □ NACK

3.2.4.2.2.Fullset [Voice Message Storage] FU command details

3.2.4.2.2.1.CreateFolderVM

Client	**Server**

CreateFolderVM =>

<= ACK(FolderID)/NACK(ReturnCode)

This Command is used to create a new folder in the Voice Message Systems device.

A user may use this command to create multiple folders for grouping of voice messages in his/her own preferable style. For example, a user may create a new folder every month for grouping of received voice messages on a monthly basis.

```
CreateFolderVM               ::= [APPLICATION tagCreateFolderVM] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    folderType                   [0] FolderType            OPTIONAL,
    descriptiveComment           [1] DescriptiveComment    OPTIONAL,
    accessMode                   [2] AccessMode            OPTIONAL
                                     -- creator specifies accesses allowed to other users
}

FolderType                   ::= ENUMERATED
{
    draft                        (0),
    deleted                      (1),
    filed                        (2),
    inbox                        (3),
    outbox                           (4),
    sent                         (5)
}

AccessMode                   ::= ENUMERATED
{
    readOnly                     (1),
    readWrite                    (2),
    other                        (127)
}
```

ACK Response

FolderID of the newly created folder (FolderID ::= INTEGER)

NACK Response

Name	Description	ReturnCode
rcInvalidFolderType	Specified folder type is not valid	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidAccessMode	Specified access mode is not valid	
rcInvalidDescriptiveComment	Specified descriptive comment is not valid	
rcVmsAccessRejected	Access to Voice Message Systems device has not been authorized	

3.2.4.2.2.2.DeleteFolderVM

Client	Server
DeleteFolderVM =>	
	<= **ACK(NULL)/NACK(ReturnCode)**

This command is used to delete a folder in the Voice Message Systems device.

For example, a user, who maintains a separate folder for each month's voice messages, may like to keep only voice messages received in the past one year and not before that. For this, the user can use the DeleteVM command (described later) to delete the voice messages in a folder that is older than a year, and can then use the DeleteFolderVM command to delete the folder itself.

```
DeleteFolderVM                ::= [APPLICATION tagDeleteFolderVM] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
   folderId                     [0]FolderID
}
```

ACK Response

NULL

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcFolderNotEmpty	Specified folder is not empty	

3.2.4.2.2.3.ChangeFolderDescVM

Client	Server

ChangeFolderDescVM =>

<= ACK(NULL)/NACK(ReturnCode)

This command is used for changing the descriptive comment of an already existing folder.

A user may use this command for changing the descriptive comment of a folder when the folder is used for storing new types of voice messages that were not planned to be stored in it when the folder was created.

```
ChangeFolderDescVM            ::= [APPLICATION tagChangeFolderDescVM] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
    folderId                      [0] FolderID,
    descriptiveComment            [1] DescriptiveComment
}
```

ACK Response

NULL

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidDescriptiveComment	Specified descriptive comment is not valid	

3.2.4.2.2.4.StoreVM

Client	Server

StoreVM =>

<= ACK(VoiceMsgID)/NACK(ReturnCode)

This command is used to store a voice message into a folder of the Voice Message Systems device via data channel. The voice message to be stored must be available in a file (already stored in a file by some mechanism outside the scope of Salutation).

A user may use this command to create a new voice message in a folder, and then use the SendVM command to send it to another user. Another use of this command is given in the description of ConcatenateVM command described below.

```
StoreVM                         ::= [APPLICATION tagStoreVM] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
    folderId                         [0] FolderID,
    dataHandle                       [1] DataHandle,
    vmsInfo                          [2] VMSInfo              OPTIONAL,
    dataTransferMode                 [3] DataTransferMode     OPTIONAL,
    voiceMessageDataDescriptor       [4] VoiceMessageDataDescriptor      OPTIONAL,
    descriptiveComment               [5] DescriptiveComment   OPTIONAL
}

DataHandle                      ::= INTEGER

VMSInfo                         ::= SEQUENCE
{
    name                             [0] DisplayStringOPTIONAL,
    section                          [1] DisplayStringOPTIONAL,
    company                          [2] DisplayStringOPTIONAL,
    phoneNumber                      [3] TelephoneNumberString           OPTIONAL,
    faxNumber                        [4] TelephoneNumberString           OPTIONAL,
    address                          [5] DisplayStringOPTIONAL,
    subject                              [6] DisplayString        OPTIONAL
}

DataTransferMode                ::= ENUMERATED
{
    immediate                        (0),
    delayed                          (1)
}
```

ACK Response

VoiceMsgID of the stored voice message.

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidDataHandle	Specified dataHandle is not valid	
rcInvalidDataTransferMode	Specified dataTransferMode is not valid	
rcInvalidEncodingAlgo	Specified encoding algorithm not valid	
rcInvalidSamplingRate	Specified sampling rate not valid	
rcInvalidVoiceMessageDescriptor	Specified VoiceMessageDescriptor not valid	
rcInvalidVoiceMessageDataFormat	Specified VoiceMessageDataFormat not valid	

rcInvalidVoiceMessageFormatInterpretation	Specified VoiceMessageFormatInterpretation not valid	
rcStorageFull	Physical storage is full	

3.2.4.2.2.5.RecordVM

Client	**Server**

RecordVM =>

<= ACK(VoiceMsgID, JobHandle)/NACK(ReturnCode)

This command is used to start recording a voice message from a telephone in a specified folder.

A user may use this command in similar situations as that for StoreVM, except that the voice message is recorded by using voice channel instead of data channel.

```
RecordVM                          ::= [APPLICATION tagRecordVM] SEQUENCE
{
                                      COMPONENTS OF MsgHeader,
    folderId                          [0] FolderID,
    telephoneNo                       [1] TelephoneNumberString,
    vmsInfo                           [2] VMSInfo              OPTIONAL,
    maxDuration                       [3] INTEGER              OPTIONAL,
    overwriteMode                     [4] BOOLEAN              OPTIONAL,
    voiceLevel                        [5] INTEGER              OPTIONAL,
    voiceMessageDataDescriptor        [6] VoiceMessageDataDescriptor       OPTIONAL,
    descriptiveComment                [7] DescriptiveComment   OPTIONAL
}
```

ACK Response

VoiceMsgID of the recorded voice message, and jobHandle.

NACK Response

Name	**Description**	**ReturnCode**
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidTelephoneNo	Specified telephone number is not valid	
rcInvalidMaxDuration	Specified maxDuration is not valid	
rcInvalidVoiceLevel	Specified voiceLevel is not valid	
rcInvalidEncodingAlgo	Specified encoding algorithm not valid	

rcInvalidSamplingRate	Specified sampling rate not valid	
rcInvalidVoiceMessageDescriptor	Specified VoiceMessageDescriptor not valid	
rcInvalidVoiceMessageDataFormat	Specified VoiceMessageDataFormat not valid	
rcInvalidVoiceMessageFormatInterpretation	Specified VoiceMessageFormatInterpretation not valid	
rcStorageFull	Physical storage is full	

3.2.4.2.2.6.RetrieveVM

Client	**Server**

RetrieveVM =>

<= **ACK(NULL)/NACK(ReturnCode)**

This command is used to retrieve a specified voice message stored in a specified folder of the Voice Message Systems device. The voice message is transported via data channel and played on the speaker of a PC/WS.

A user may use this command to listen to a received voice message, or to verify the contents of a voice message that he/she has created using the StoreVM command.

```
RetrieveVM                        ::= [APPLICATION tagRetrieveVM] SEQUENCE
{
                                      COMPONENTS OF MsgHeader,
    folderId                          [0] FolderID,
    voiceMsgId                        [1] VoiceMsgID,
    headerInformation                 [2] BOOLEAN              OPTIONAL,
    voiceMessageDataDescriptor        [3] VoiceMessageDataDescriptor     OPTIONAL,
    voiceDuration                     [4] INTEGER              OPTIONAL,
    voiceSpeed                        [5] INTEGER              OPTIONAL,
    voiceVolume                       [6] INTEGER              OPTIONAL
}
```

ACK Response

NULL

NACK Response

Name	**Description**	**ReturnCode**
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidVoiceMsgId	Specified Voice Message not found	

rcInvalidHeaderInfo	Specified headerInformation is not valid	
rcInvalidEncodingAlgo	Specified encoding algorithm not valid	
rcInvalidSamplingRate	Specified sampling rate not valid	
rcInvalidVoiceMessageDescriptor	Specified VoiceMessageDescriptor not valid	
rcInvalidVoiceMessageDataFormat	Specified VoiceMessageDataFormat not valid	
rcInvalidVoiceMessageFormatInterpretation	Specified VoiceMessageFormatInterpretation not valid	
rcInvalidVoiceDuration	Specified voiceDuration is not valid	
rcInvalidSpeed	Specified voiceSpeed is not valid	
rcInvalidVoiceVolume	Specified voiceVolume is not valid	

3.2.4.2.2.7.SetReceiverOptionsVM

Client	**Server**

SetReceiverOptionsVM =>

<= **ACK(NULL)/NACK(ReturnCode)**

This command is used to allow a receiver to set options that tell the Voice Message Systems how the messages received for this receiver are to be treated.

A user may use this command, for example, to tell the Voice Message Systems device about a forwarding address where he/she wants all his/her voice messages to be forwarded.

```
SetReceiverOptionsVM          ::= [APPLICATION tagSetReceiverOptionsVM] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
    holdForDelivery               [0] HoldForCriteria          OPTIONAL,
    implicitConversion            [1] BOOLEAN                  OPTIONAL,
    redirectionAddress            [2] Recipient                OPTIONAL,
    restrictedDeliveryId          [3] SET OF Recipient         OPTIONAL,
    storedMsgAlert                [4] BOOLEAN                  OPTIONAL,
    autoForwardAddress            [5] Recipient                OPTIONAL
}

HoldForCriteria               ::= SEQUENCE
{
                                  -- for further study
}
```

ACK Response

NULL

NACK Response

Name	Description	ReturnCode
rcInvalidHoldForDelivery	Specified holdForDelivery criteria is not valid	
rcInvalidRedirectionAddress	Specified redirection address is not valid	
rcInvalidRestrictedDeliveryId	Specified restrictedDeliveryId is not valid	
rcInvalidForwardAddress	Specified forward Address is not valid	

3.2.4.2.2.8.DeleteVM

Client	Server

DeleteVM =>

<= ACK(NULL)/NACK(ReturnCode)

This command is used to delete a voice message from a folder in the Voice Message Systems device.

A user may use this command to delete a voice message that is no more needed in a folder. Some other uses of this command are given during the description of DeleteFolderVM, and SeparateVM commands.

```
DeleteVM                          ::= [APPLICATION tagDeleteVM] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
    folderId                      [0] FolderID,
    voiceMsgId                    [1] VoiceMsgID
}
```

ACK Response

NULL

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidVoiceMsgId	Specified Voice Message not found	

3.2.4.2.2.9.CopyVM

Client	Server

CopyVM =>

<= ACK(VoiceMsgID)/NACK(ReturnCode)

This command is used to make a copy a specified voice message in the same folder. When operation is completed, the VoiceMsgID assigned to the copied message is returned.

For example, a user wants to slightly modify an existing voice message before sending it to another user, but also wants that the original voice message be left unchanged for later use. He/she can achieve this by first making a copy of the existing voice message by using this command, then using one/more of the voice message manipulation commands (described later) to edit the copy, and finally using the SendVM command to send the modified version of the voice message to the desired receiver(s).

```
CopyVM                          ::= [APPLICATION tagCopyVM] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    folderId                        [0] FolderID,
    voiceMsgId                      [1] VoiceMsgID
}
```

ACK Response

VoiceMsgID of copies voice message

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidVoiceMsgId	Specified Voice Message not found	
rcStorageFull	Physical storage is full	

3.2.4.2.2.10.ConcatenateVM

Client	**Server**

ConcatenateVM =>

<= ACK(VoiceMsgID)/NACK(ReturnCode)

This command is used to combine multiple voice messages stored in the Voice Message Systems device, in the sequence provided, into a single resulting voice message. When the operation completes, the VoiceMsgID assigned to the resulting voice message is returned.

For example, a user receives a voice message from another user. He/she wants to forward it to another user by prepending/appending his/her own voice comments to it. To do this, he/she can create a voice message for his/her own comments by using the StoreVM command, then use the ConcatenateVM command to prepend/append it to the received voice message, and finally use the SendVM command to forward it. Another use of this command is given in the description of SeparateVM command described below.

```
ConcatenateVM                   ::= [APPLICATION tagConcatenateVM] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    outputFolderId                  [0] FolderID,
    inputVoiceMsg                   [1] InputVoiceMsg
}

InputVoiceMsg                   ::= SET OF SEQUENCE
{
    folderId                        [0] FolderID,
    voiceMsgId                      [1] VoiceMsgID
}
```

ACK Response

VoiceMsgID of the resulting voice message

NACK Response

Name	**Description**	**ReturnCode**
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidVoiceMsgId	Specified Voice Message not found	
rcStorageFull	Physical storage is full	

3.2.4.2.2.11.SeparateVM

Client	Server

SeparateVM =>

<= ACK(VoiceMsgIDs)/NACK(ReturnCode)

This command is used to divide a specified voice message of the Voice Message Systems device into two voice messages. The resulting voice messages are stored in the same folder as that of the original voice message. The original voice message is left unchanged and VoiceMsgIDs of resulting voice messages is returned.

This command may be used by a user for creating two voice messages out of an already existing voice message. It may also be used by a user along with DeleteVM and ConcatenateVM commands to delete a part of an existing voice message. Only SeparateVM and DeleteVM are needed when one end of an existing voice message is to be deleted. If a portion that does not belong to one end of an existing voice message is to be deleted, then SeparateVM, DeleteVM, and ConcatenateVM commands are needed.

```
SeparateVM                  ::= [APPLICATION tagSeparateVM] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   folderId                     [0] FolderID,
   voiceMsgId                   [1] VoiceMsgID,
   position                     [2] INTEGER
}
```

ACK Response

Two VoiceMsgIDs of the resulting voice message

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	
rcFolderAccessRejected	Access to folder has not been authorized	
rcInvalidVoiceMsgId	Specified Voice Message not found	
rcInvalidPosition	Specified position is not valid	
rcStorageFull	Physical storage is full	

3.2.4.2.2.12.RepositionVM

Client	Server

RepositionVM =>

<= ACK(CurrentPosition)/NACK(ReturnCode)

This command is used to move the current pointer position forward/backward by a specified period in a voice message. When operation is complete, current position is returned.

A user wants to listen to only a portion of a voice message somewhere from the middle. The user can achieve this by using RepositionVM and PlayVM commands.

```
RepositionVM                ::= [APPLICATION tagRepositionVM] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    voiceMsg                    [0] VoiceMsg,
    period                      [1] INTEGER
}

VoiceMsg                    ::= CHOICE
{
    voiceMsgId                  [0] VoiceMsgID,
                                -- before starting play
    jobHandle                   [1] JobHandle
                                -- after starting play
}
```

ACK Response

CurrentPosition (::= INTEGER)

NACK Response

Name	Description	ReturnCode
rcInvalidVoiceMsgId	Specified Voice Message not found	
rcInvalidJobHandle	Specified jobHandle is not valid	
rcInvalidPeriod	Specified period is not valid	

3.2.4.2.2.13.ReviewVM

Client	**Server**
ReviewVM =>	
	<= ACK(NULL)/NACK(ReturnCode)

This command is used to play a portion of a voice message during a record session.

During a recording session (RecordVM command execution), a user wants to listen to a portion of the voice message being recorded to verify its contents before continuing to record. He/she can do this by using the ReviewVM command.

```
ReviewVM                    ::= [APPLICATION tagReviewVM] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   jobHandle                    [0] JobHandle,
   period                       [1] INTEGER
}
```

ACK Response

NULL

NACK Response

Name	**Description**	**ReturnCode**
rcInvalidJobHandle	Specified jobHandle is not valid	
rcInvalidPeriod	Specified period is not valid	

4.Personal Information Systems

4.1.Personal Information Systems Overview

This section describes the Functional Unit that provides services associated with personal information. The following Functional Unit is defined in version 2 to handle respective personal information.

- [Address Book]

[Address Book] Functional Unit allows a client application to access and manipulate its address book information.

A typical equipment that has [Address Book] Functional Unit is as follows:

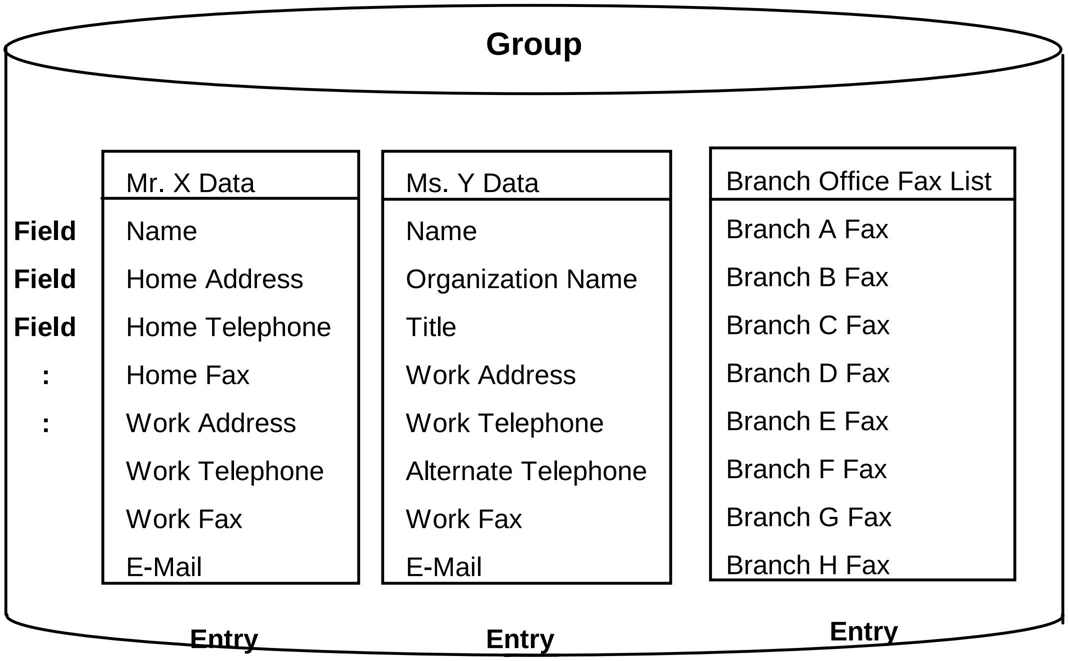
- PDA (Personal Digital Assistant)
- Telephone or FAX that contains a user-definable telephone directory
- Personal computer with a personal information management (PIM) application

This version of the architecture focuses on the services that may be provided by such personal equipment. It does not intend to cover full services which are provided by the large "server" computer that may support corporate or work group level address book data base for many users.

4.1.1.Common Characteristics in Personal Information Systems

4.1.1.1.Concept of Group, Entry and Field

The Personal Information Systems support the concept of the Group, the Entry and the Field. The Entry is a block of personal information data. The examples of Entry are business card data of each person, distribution list of FAX mail, etc. The Field is each data in an Entry. The examples of Field are person’s Name, Address, Title, Telephone number, etc. Each Field has parameters to describe the Field more specifically. The examples of Telephone number parameter are Home, Work, Pager, etc. The Group is a container of the Entries. A Group may have multiple Entries in it.



An FU maintains Groups In It and a client can retrieve personal data In an FU.

Each Group, or Entry is identified by the Group Handle and the Entry Handle which are assigned by an FU. Each Field is identified by the Field Name which is uniquely defined by the architecture.

4.1.1.2.Exchange data format

Salutation Architecture defines the exchange data format for the Group and the Entry when it is exchanged between an FU and a client, however, how an FU or a client has personal information data in it is an implementation option, i.e., an FU or a client can have personal information data in any format. A client will specify the returned data format when it requests to get Group data or Entry data from an FU. A client will also specify the data format when it puts personal data into an FU.

Supported data format by an FU will be informed by the capability attribute. An FU will accept the supported data format.

In version 2, versit's vCard is supported as an exchange data format. Refer to versit Electronic Business Card (vCard) Specification for vCard definition in detail.

4.1.1.3.Data encoding for coded personal data

When exchange data is coded personal data like name, address, etc., it is encoded to be expressed in 8 bits or in 7 bits. Supported encoding for coded data by an FU will be informed by the capability attribute. An FU will accept coded data which is encoded by the supported encoding.

A client will specify the encoding for returned coded data when it requests to get personal data from an FU. A client will not specify the encoding for coded data when it puts personal data into an FU because encoding is specified in the data.

4.1.1.4.Character set encoding for coded personal data

Supported character set for coded personal data by an FU will be informed by the capability attribute. An FU will accept coded personal data which is encoded by the supported character set.

To identify the character set for exchange data, an FU or a client will set the character set In each Entry.

4.1.1.5.Data encoding for binary personal data

When exchange data is binary personal data like pronunciation of name (audio data), or logo (image data), it is encoded to be expressed in 8 bits or in 7 bits. Supported encoding for binary data by an FU will be informed by the capability attribute. An FU will accept binary data which is encoded by the supported encoding.

A client will specify the encoding for returned binary data when it requests to get personal data from an FU. A client will not specify the encoding for binary data when it puts personal data into an FU because encoding is specified in the data.

4.1.1.6.Operations for Group, Entry and Field

4.1.1.6.1.Group Operation

To get all Group names in an FU, ListGroups command is used. When this command is issued, list of all Group names and their access mode, Read only or write access, will be returned to a client. A client will use the OpenGroup command to open the Group. The Group name and the access mode will be specified to open the Group. When the Group is opened, an FU will assign the Group Handle to the Group and return it to a client with an ACK. Group Handle is the unique value to identify the Group in an FU and valid until the Group is closed. Multiple Groups can be opened independently. To close the Group, CloseGroup command is used.

The operations for the Groups are Create a Group, Delete a Group, Rename Group name and Get Group Data. CreateGroup command is used to create a new Group. Group name is passed to an FU and an FU returns the Group Handle with an ACK for further operation. Put data into the Group will be performed by another command, like AddEntryData because CreateGroup will be used only to create a personal data container. DeleteGroup command is used to delete a Group. Only the Group which has no Entry in it can be deleted. If the Group has an Entry in it, the request will be rejected. RenameGroup command is used to rename the Group name. The new name is passed to an FU, but the Group Handle assigned to the Group is not changed. GetGroupData command is used to get whole data in the Group. Returned data format and encoding for coded data and binary data will be specified by a client. Create a Group, Delete Group and Rename Group name operations are permitted to a client who can access to the Group in write access mode.

4.1.1.6.2.Entry Operation

Personal Information Systems provide the Field data search operation as an optional function. When specified Field data is found in an Entry, this Entry is marked as an **Active Entry**, then next search operation continues. When the search operation completes, an FU assigns the Entry Handles to those Active Entries dynamically to identify each Entry in the Group. Entry Handles are valid until next search operation is performed. If an FU does not support the search function, Entry related operations can not be supported because Entry operations require Entry Handle to identify the Entry.

Operations for Entries are List Active Entries, Get Entry Data, Get Active Entry Data, Add Entry Data, Delete Entry Data, Replace Entry Data, Move Entry Data and Copy Entry Data.

ListActiveEntries command is used to get a list of Active Entries. The set of Group Handle, Entry Handle and character set of each Active Entry will be returned to a client. GetEntryData command is used to get an active Entry data in a Group. The Group Handle and the Entry Handle which are returned to the ListActiveEntries command will be used to specify the Entry. The data format and encoding for coded data and binary data of the returned personal data will be specified by a client. GetActiveEntryData command is also used to get an active Entry data. The difference from GetEntryData command is, this command uses the position to specify the Entry in the Active Entries. The position starts from one to identify the first Entry in the active Entries. The data format and encoding for coded data and binary data of returned personal data will be also specified by a client. AddEntryData is used to add Entry data into a Group. ‘To’ Group will be specified by the Group Handle. DeleteEntryData command is used to delete Entry data in a Group. The Group Handle and the Entry Handle are used to specify the Entry. ReplaceEntryData command is used to replace Entry data in a Group. To specify the Entry, Group Handle and the Entry Handle are used. MoveEntryData command is used to move the Entry data to another Group. To specify the Entry, the Group Handle and the Entry Handle are used. To specify the ‘to’ Group, ‘to’ Group Handle is used. An FU will return the new Entry Handle with ACK. CopyEntryData command is used to copy the Entry data. To specify the Entry, the Group Handle and the Entry Handle are used. ‘To’ Group is specified by the Group Handle. An FU will return the new Entry Handle with ACK.

Add Entry Data, Delete Entry Data, Replace Entry Data, Move Entry Data and Copy Entry Data operations are permitted for a client who can access to the Group in write access mode.

4.1.1.6.3.Field Operation

The Field in each Entry is identified by the Field Name, which is uniquely defined by the architecture. An Entry may have multiple same Field data in it.

Operations for Fields are Search Field Data and Get Active Entries Field Data. Refer to next paragraph for Field search operation. GetActiveEntriesFieldData command is used to get specific Field data in Active Entries. Field Name is used to specify the Field in an Entry. A set of Group Handle, Entry Handle and Field data value in Active Entries will be returned. If an Entry has a hierarchy structured data, or multiple Fields data, all Field data specified by Field Name will be returned. An FU will sort the Field data before it returns a set of Fields data to a client. The sort is an optional function.

4.1.1.6.4.Field Data Search Operation

An FU supports the Field data search operation as an optional function. SearchFieldData command is used to search specific Field data in an Entry. The search operation will be performed only for string data in an Entry which data is encoded by the specified character set. To specify the Field, ‘Field Name’ or ‘ALL’ is used. When Field Name is specified, the data of the specified Field will be searched. When ‘ALL’ is specified, all Fields data in an Entry will be searched. Example of ‘ALL’ search operation is when ‘ALL=New York’ is specified, an FU will search the value of ‘New York’ in all Fields, i.e. Name fields, Address fields, Telephone number fields, Title fields, etc. When Field has parameter(s), Field Name and parameter(s) will be used to search specific Field data.

When Field parameter(s) is (are) specified, an FU will search all combinations of Field and parameter(s) which match to the specified Field Name and parameter(s). Example of the Field Name with parameter search is Telephone number with HOME parameter. In this case, an FU will search all Fields data which Field is TEL with HOME parameter, they are TEL (HOME, FAX), TEL (HOME, VOICE), TEL (HOME, MGS, VOICE), TEL (CELL, HOME, WORK), etc. Therefore, if a client does not specify the parameter for the Field, an FU will search all specified Fields which have any parameters. Field Name and its parameters are always encoded by 8859-1 (US ASCII) character set.

The search operation is simple Field data comparison with the specified value, if it is ‘Equal to’, ‘Greater than’ or ‘Less than’ the specified value. The ‘Equal to’ comparison will take place from the top of the Field by shifting one by one character data if it is equal to the specified value. ’Greater than’ or ‘Less than’ is valid for numeric data like Telephone numbers. From the top of the Field, whole data is compared with the specified value. When Telephone number is compared, non-numeric numbers are treated as null and not used for the comparison. Example of this case is, ‘(919)254-1111’, ‘919-254-1111’, ‘919 254 1111’ are treated as ‘9192541111’.

The logical operation, ‘AND’ or ‘OR’ will be supported for the comparison. An example of ‘OR’ logical operation is to find the name of ‘Goldsmith’, ‘GOLDSMITH’ or ‘goldsmith’. An example of ‘AND’ operation is to find the name of ‘Goldsmith’ whose address is ‘New York’.

Also two levels of ‘Wildcard’ character data search operation are supported. The asterisk (*) is used for variable length string data pattern matching and the question (?) is for one string data pattern matching. An example of ‘*’ is to find the family name of Goldsmith, Silversmith, Blacksmith or Coppersmith by specifying ‘*smith’. An example of ‘?’ is to find the telephone number whose area code is specific, for example by specifying ‘919254????’.

When Field data meets the search condition, the Entry is marked as an Active Entry. When an Entry has hierarchy structured for personal data, all Field data specified by Field Name will be compared with the specified value.

Two types of Search operation are supported. The first type search is to search specific Field data in *ALL Entries* in the specified Groups. When Field is found in an Entry which meets the search condition, this Entry is marked as an Active Entry, then search operation is continued until all remained Entries are searched. This operation is specified by setting the Search Handle value to zero. The second type search is to search specific Field data in the *CURRENT Active Entries.* This operation is specified by setting the Search Handle to the value which is returned to the previous search operation. This allows a client to narrow down the target Entry which has the specific data a client wants to get. In both cases, an FU returns the number of found Active Entries and the Search Handle.

4.1.1.7.List of Messages in Personal Information Systems

The following commands and responses are used in the Personal Information Systems Functional Unit.

- Personal Information Systems Mandatory support Commands;
  - □ ListGroups
  - □ OpenGroup
  - □ CloseGroup
  - □ GetGroupData

- Personal Information Systems Mandatory support Common Commands and Responses;
  - □ RequestDataTransfer
  - □ DataBlockDescription
  - □ TransferDataBlock
  - □ RequestNextData
  - □ ACK and NACK
- Personal Information Systems Optional support Commands; (These commands belong to the same optional support group, so all these commands should be supported if an FU support them.)
  - □ CreateGroup
  - □ DeleteGroup
  - □ RenameGroup
  - □ ListActiveEntries
  - □ GetEntryData
  - □ GetActiveEntryData
  - □ AddEntryData
  - □ DeleteEntryData
  - □ ReplaceEntryData
  - □ MoveEntryData
  - □ CopyEntryData
  - □ SearchFieldData
  - □ GetActiveEntriesFieldData
- Personal Information Systems Optional support Command
  - □ VendorEscape

4.2.[Address Book] Functional Unit

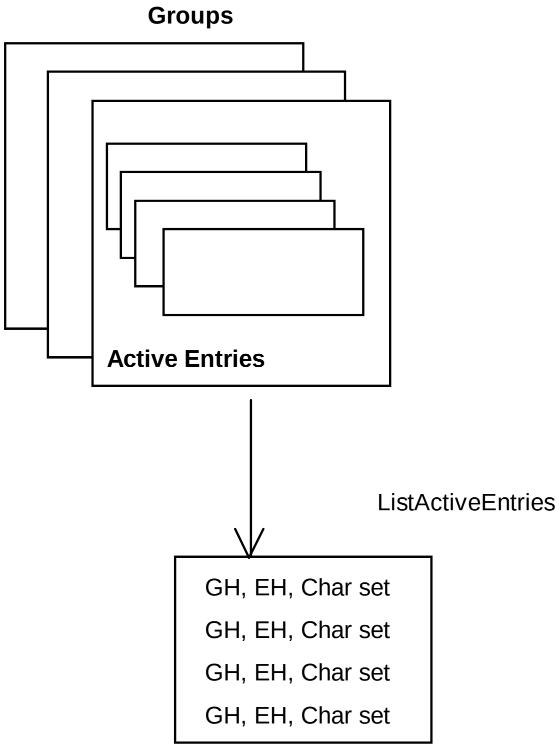
4.2.1.Overview

An [Address Book] Functional Unit maintains address book information in it.

Commands used for the [Address Book] Functional Unit and their usage examples are as follows;

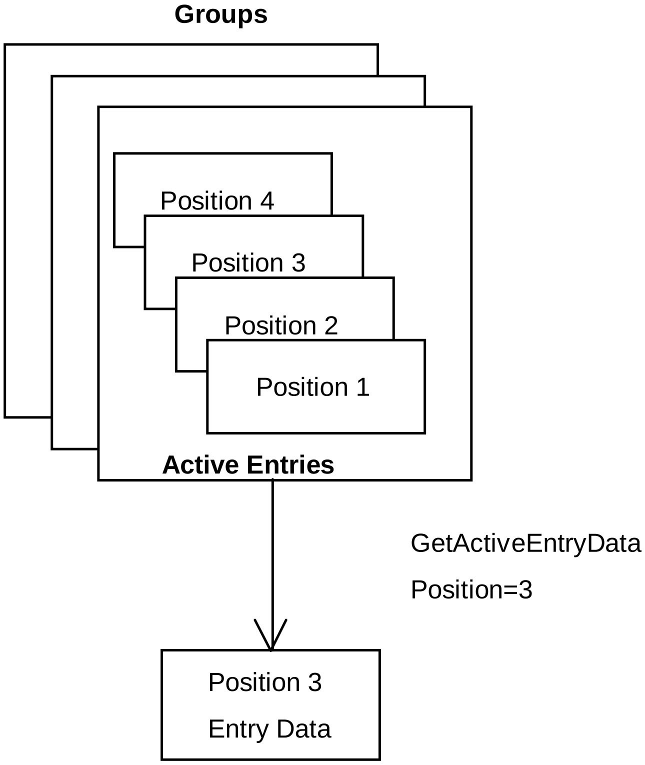
- **ListGroups**, which returns a list of all Group Names and their access modes.
- **OpenGroup**, which opens specific Group and returns Group Handle.
- **CloseGroup**, which closes the Group.
- **CreateGroup**, which creates a new Group as a data container and returns its Group Handle.
- **DeleteGroup**, which deletes the Group.

- **RenameGroup**, which changes the Group name.
- **GetGroupData,** which returns whole Group data.
- **ListActiveEntries**, which returns a set of Group Handle (GH), Entry Handle (EH) and character set of Active Entries.



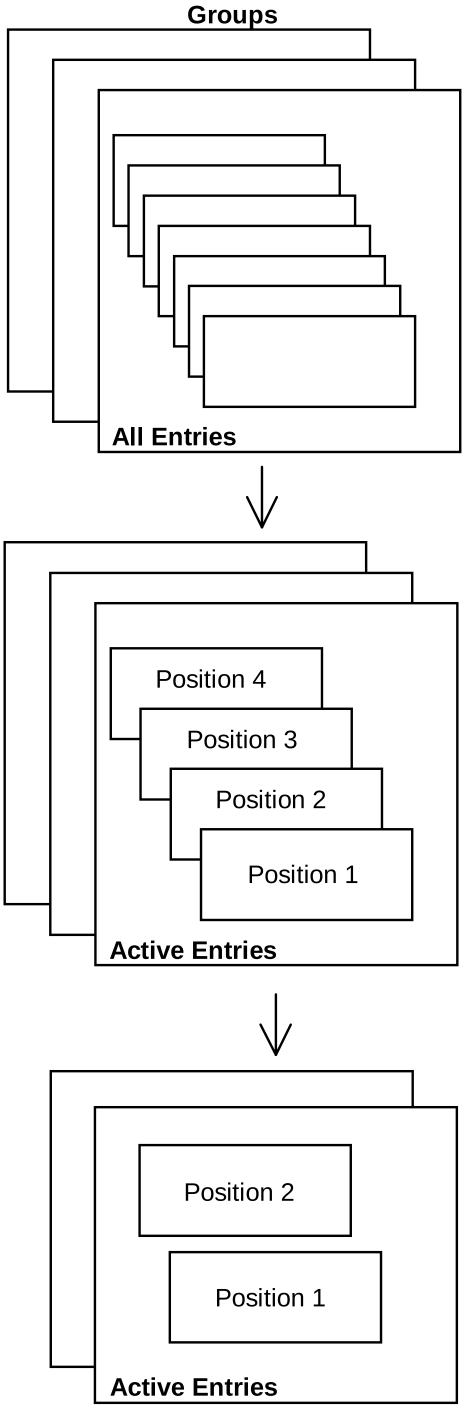
- **GetEntryData**, which returns whole Active Entry data which is specified by the Group handle and the Entry handle.

- **GetActiveEntryData**, which returns whole Active Entry data which is specified by the position in Active Entries.



- **AddEntryData**, which adds an Entry data into a Group.
- **DeleteEntryData**, which deletes an Entry data in a Group.
- **ReplaceEntryData**, which replaces existing Entry data with new Entry data.
- **MoveEntryData**, which moves Entry data to another Group.
- **CopyEntryData**, which copies Entry data to a Group as another Entry data and returns an Entry Handle.

- **SearchFieldData**, which searches the Field data that meets the specified search condition, and returns the number of found Active Entries.



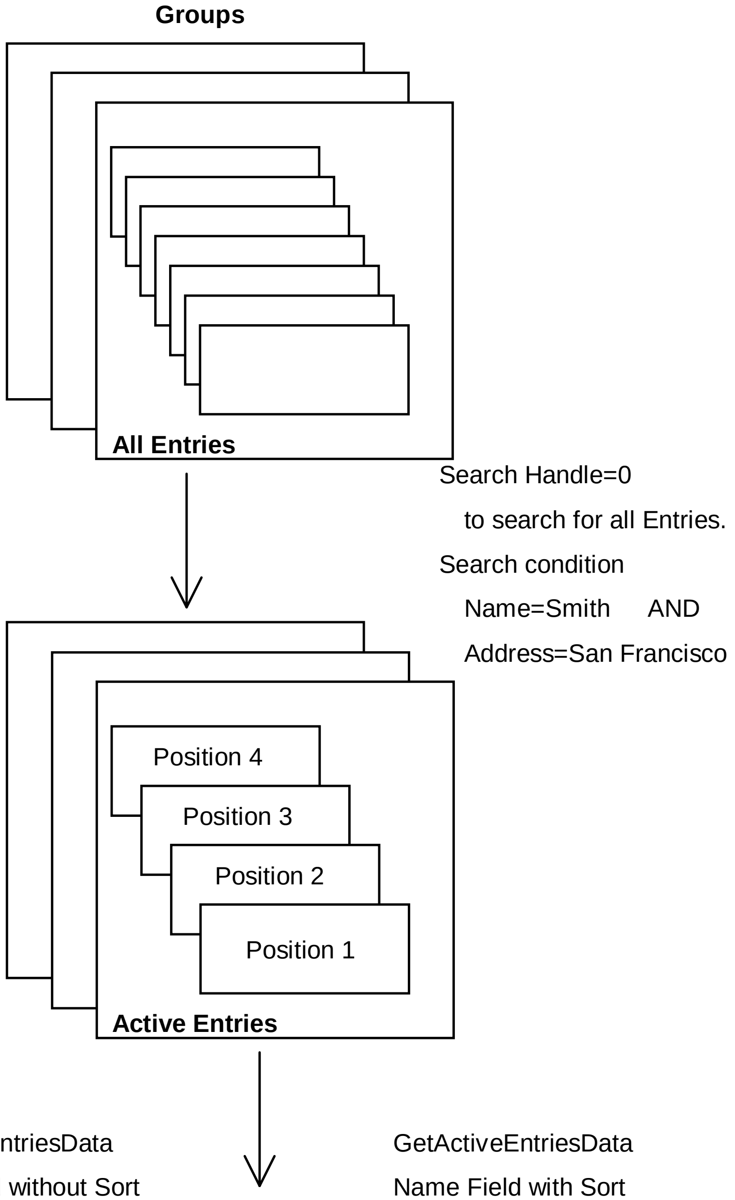
Search Handle=0

to search for all Entries.

Search Handle=previous Handle to

search for current Active Entries.

- **GetActiveEntriesFieldData**, which returns a set of Group Handle (GH), Entry Handle (EH) and specified Field data. The Field data can be sorted before they are returned to a client.

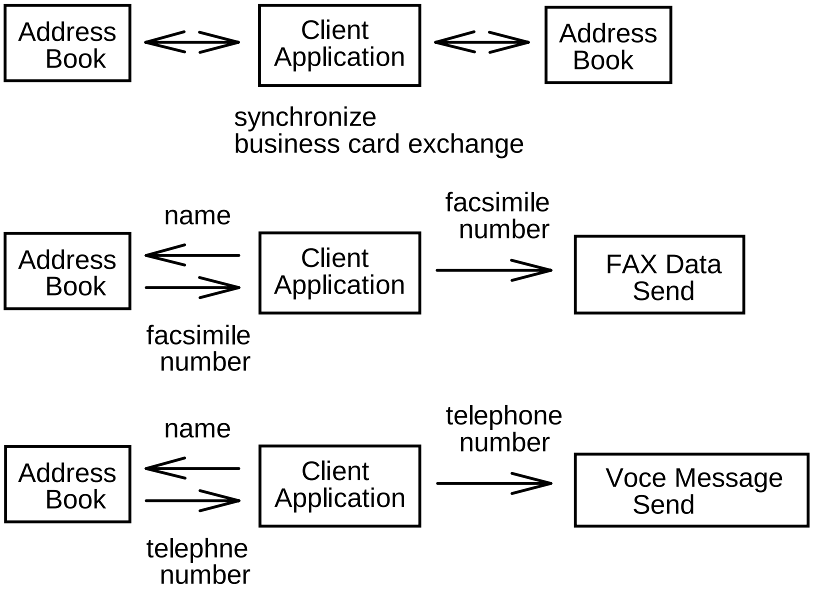


GetActiveEntriesData Name Field without Sort		GetActiveEntriesData Name Field with Sort	
GH, EH, Smith, Mike	(Position 1)	GH, EH, Smith, Anne M	(Position 13)
GH, EH, Smith, Barry	(Position 2)	GH, EH, Smith, Barry	(Position 2)
GH, EH, Smith, Ken	(Position 3)	GH, EH, Smith, Bertha L	(Position 8)
GH, EH, Smith, Kevin	(Position 4)	GH, EH, Smith, Billy	(Position 24)
:		:	

When target person is found in this list, a client will use Group handle and Entry handle to get whole Entry data.

This version of the architecture focuses on the personal use, so there are restrictions as follows:

- There is no direct interaction between two Functional Units. Namely, a client application must interact with two Functional Units to exchange data between them. For instance, a client application is responsible for synchronizing two address books in two PDAs.



- There is no Personal Information Systems unique security control. How to define Read only or Write access for the Group is out of the scope.

4.2.2.Examples of operational sequence

4.2.2.1.To get whole Group data

A typical sequence of commands to get whole Group data is as follows:

1) Use ListGroups command to know all Group names.
2) Use OpenGroup command to open the target Group and to get Group handle.
3) Use GetGroupData command to get whole Group data in the specified format.
4) Use CloseGroup command to close the Group.

4.2.2.2.To get searched Entry data by specifying the Entry

A typical sequence of commands to search and get Entry data in a Group by specifying the handles is as follows:

1) Use ListGroups command to get all Group names.
2) Use OpenGroup command to open the target Group(s) and to get Group handle(s).

3) Use SearchFieldData command to search specific Field data and to get the number of active Entries. Continue this operation until the number of active Entries becomes small.

4) Use ListActiveEntries command to get a set of Group handle and Entry handle.

5) Use GetEntryData command to get Entry data by specifying the handles returned to ListActiveEntries command. Continue this operation until the target Entry data is found.

6) Use CloseGroup command to close the Group.

4.2.2.3.To get searched Entry data by specifying the position

A typical sequence of commands to search and get Entry data in a Group by specifying the position is as follows:

1) Use ListGroups command to get all Group names.

2) Use OpenGroup command to open the target Group(s) and to get Group handle(s).

3) Use SearchFieldData command to search specific Field data and to get the number of active Entries. Continue this operation until the number of active Entries becomes small.

4) Use GetActiveEntryData command to get Entry data by specifying the position in active Entries. Continue this operation until the target Entry data is found.

5) Use CloseGroup command to close the Group.

4.2.2.4.To get Entry data by getting Active Entries Field Data

A typical sequence of commands to get an Entry data by getting Active Entries Field Data is as follows:

1) Use ListGroups command to get all Group names.

2) Use OpenGroup command to open the target Group(s) and to get Group handle(s).

3) Use SearchFieldData command to search specific Field data and to get the number of active Entries. Continue this operation until the number of active Entries becomes small.

4) Use GetActiveEntriesFieldData command to get all active Entries Field data. The specified Field may be different from the Field which is used for search operation. If an FU supports the sort function and the sort is specified, returned Field data will be sorted. Continue this operation until target Entry data is found and its Group handle and Entry handle is gotten.

5) Use GetEntryData command to get Entry data by specifying the Group handle and Entry handle.

6) Use CloseGroup command to close the Group.

4.2.3.Field Names of [Address Book] FU

[Address Book] FU adopts the Field Name from versit’s vCard Property name. Following is the examples of supported Field Names. Refer to versit Electronic Business card (vCard) Specification for details.

In addition to these Field Names, [Address Book] FU supports ‘ALL’ as a Field Name for search operation.

Field	**Field Name**	**Parameters**
Formatted Name	FN	-
Name	N	-
Photograph	PHOTO	GIF, CGM, WMF, BMP, MET, PMB, DIB, PICT, TIFF, PS, PDF, JPEG, MPEG, MPEG2, AVI, QTIME
Birth date	BDAY	-
Address	ADR	DOM, INTL, POSTAL, PARCEL, HOME, WORK
Address delivery label	LABEL	DOM, INTL, POSTAL, PARCEL, HOME, WORK
Telephone number	TEL	PREF, WORK, HOME, VOICE, FAX, MSG, CELL, PAGER, BBS, MODEM, CAR, ISDN, VIDEO
Electronic mail	EMAIL	AOL, AppleLink, ATTMail, CIS, eWorld, INTERNET, IBMMail, MCIMail, POWERSHARE, PRODIGY, TLX, X400
Mailer	MAILER	-
Time zone	TZ	-
Location	GEO	-
Title	TITLE	-
Business category	ROLE	-
Logo	LOGO	GIF, CGM, WMF, BMP, MET, PMB, DIB, PICT, TIFF, PS, PDF, JPEG, MPEG, MPEG2, AVI, QTIME
Organization	ORG	-
Comment	NOTE	-
Pronunciation	SOUND	WAVE, PCM, AIFF
Uniform Resource Locator	URL	-
Unique Identifier	UID	-
Public Key	KEY	-

4.2.4.List of Functional Unit Attribute

The following table shows the attributes defined for [Address Book] Functional Unit.

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private Attribute
personalityProtocol	30000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No
supportedCommand	30001	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No
exchangeDataFormatSupport	30002	ExchangeDataFormat	Set OF ExchangeDataFormat (setIntDoesContain)	No	No
characterSetSupport	30003	CharSetID	SET OF CharSetID (setIntDoesContain)	No	No
searchSupport	30004	N/A	SearchSupport (boolEqualTo)	No	No
sortSupport	30005	N/A	SortSupport (boolEqualTo)	No	No

4.2.5.Salutation Personality Message & Protocol

4.2.5.1.ListGroups command

Client	**Server**
ListGroups =>	
	<= TransferDataBlock (GroupList)
ACK (NULL) =>	

This command is used to get a list of Group names in an FU. The set of Group name and its access mode will be returned. When the command can not be handled by an FU, NACK will be returned.

```
ListGroups                  ::= [APPLICATION tagListGroups] SEQUENCE
{
                                COMPONENTS OF MsgHeader
}
```

Data transferred by TransferDataBlock command is as follows:

```
GroupList                   ::= SET OF GroupDescription

GroupDescription            ::= SEQUENCE
{
    groupName                   [0] DisplayString,
    writable                    [1] BOOLEAN                 DEFAULT FALSE
                                    -- TRUE shows Group is writable
}
```

4.2.5.2.OpenGroup command

Client	**Server**

OpenGroup =>

<= ACK (GroupHandle/ReturnCode)/NACK (ReturnCode)

This command is used to open a Group and should be issued prior to the use of a Group. Group name is used to specify the Group. Access mode is also set in the command. When the Group is ready for a client use, an FU will return a Group handle with ACK. The Group can be opened by multiple clients simultaneously for read operation. If the Group is already opened for the read/write operation, next OpenGroup request for read/write operation will be rejected. If the next OpenGroup request is for read only operation, the request will be accepted and the second parameter of ACK will show that the Group is being opened for read/write operation by another client. When the command can not be handled, NACK will be returned.

```
OpenGroup                    ::= [APPLICATION tagOpenGroup] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
   groupName                    [0] DisplayString,
   readWriteAccess              [1] BOOLEAN                    DEFAULT FALSE
                                     -- TRUE shows Writable access
}
```

ACK Response

Parameter Name	**Data Type**	**Note**
parameter-1	GroupHandle	
parameter-2	ReturnCode (rcBeingModified, ENUMERATED (148))	The Group is being opened for read/write operation by another client.

NACK Response

Name	Description	ReturnCode
rcNoGroup	There is not specified Group.	138
rcInvalidAccessMode	Access mode is not valid.	139
rcCanNotBeOpened	The Group can not be opened.	140
rcBeingModified	The Group is already opened for read/write operation	148

4.2.5.3.CloseGroup command

Client	Server

CloseGroup =>

<= ACK (NULL)/NACK (ReturnCode)

This command is used to close a Group. Group Handle is used to specify the Group. When the command can not be handled, NACK will be returned.

```
CloseGroup                    ::= [APPLICATION tagCloseGroup] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   groupHandle                    [0] INTEGER
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidGroupHandle	Specified Group Handle is invalid	141

4.2.5.4.CreateGroup command

Client	Server

CreateGroup =>

<= ACK (GroupHandle)/ NACK (ReturnCode)

This command is used to create a new Group as an Entry container. Group name is passed to an FU and an FU returns a Group Handle with ACK. When the command can not be handled, NACK will be returned.

```
CreateGroup                       ::= [APPLICATION tagCreateGroup] SEQUENCE
{
                                      COMPONENTS OF MsgHeader,
    groupName                         [0]DisplayString
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	GroupHandle	

NACK Response

Name	Description	ReturnCode
rcGroupAlreadyExist	Specified Group name already exists in an FU.	144

4.2.5.5.DeleteGroup command

Client	**Server**

DeleteGroup =>

<= **ACK (NULL)/NACK (ReturnCode)**

This command is used to delete a Group. Group Handle is used to specify the Group. If the Group has an Entry in it, NACK will be returned. This operation is permitted to a client who can access to the Group in write access mode. When the command can not be handled, NACK will be returned.

```
DeleteGroup                       ::= [APPLICATION tagDeleteGroup] SEQUENCE
{
                                      COMPONENTS OF MsgHeader,
    groupHandle                       [0] INTEGER
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcGroupHasEntry	Specified Group has an Entry.	145
rcInvalidGroupHandle	Specified Group Handle is invalid.	141

4.2.5.6.RenameGroup command

Client	Server

RenameGroup =>

<= ACK (NULL)/NACK (ReturnCode)

This command is used to rename a Group name. Group Handle is used to specify the Group. This operation is permitted to a client who can access to the Group in write access mode. The value of the Group Handle is unchanged by the rename operation. When the command can not be handled, NACK will be returned.

```
RenameGroup                ::= [APPLICATION tagRenameGroup] SEQUENCE
{
                               COMPONENTS OF MsgHeader,
   groupHandle                 [0] INTEGER,
   newName                     [1] DisplayString
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcGroupAlreadyExist	Specified Group name already exists in an FU.	144
rcOperationNotPermitted	Operation to the Group is not permitted.	146

4.2.5.7.GetGroupData command

Client	**Server**
GetGroupData =>	
	<= TransferDataBlock (GroupData)
ACK (NULL) =>	

This command is used to get whole data in the Group. Group Handle is used to specify the Group. The returned data format and encoding for coded data and binary data will be specified by a client. When the command can not be handled, NACK will be returned.

```
GetGroupData                ::= [APPLICATION tagGetGroupData] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    groupHandle                 [0] INTEGER,
    exchangeDataFormat          [1] ExchangeDataFormat
}

ExchangeDataFormat          ::= CHOICE
{
    vCard                       [0] VCardEncoding
}

VCardEncoding               ::= SEQUENCE
{
    codedEncoding               [0] CodedEncoding,
    binaryEncoding              [1] BinaryEncoding
}

CodedEncoding               ::= ENUMERATED
{
    sevenBit                    (0),
    base64Encoding              (1),
    quotedPrintable             (2),
    eightBit                    (3)
}

BinaryEncoding              ::= ENUMERATED
{
    base64Encoding              (0),
    eightBit                    (1)
}
```

Data transferred by TransferDataBlock command is as follows:

```
GroupData                   ::= SET OF EntryData
```

```
EntryData                   ::= SEQUENCE
{
    charSetID                   [0] CharSetID,
    data                        [1] OCTET STRING
                                    -- vCard format
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidExchangeDataFormat	Specified exchange data format is invalid.	168
rcExchangeDataFormatNotSupported	Specified exchange data format is not supported.	169
rcCodedEncodingNotSupported	Specified Coded encoding is not supported.	170
rcBinaryEncodingNotSupported	Specified Binary encoding is not supported.	171

4.2.5.8.ListActiveEntries command

Client	**Server**
ListActiveEntries =>	
	TransferDataBlock (ActiveEntriesList)
ACK (NULL) =>	

This command is used to get an Active Entries list which is created by the SearchFieldData operation. The set of the Group Handle , the Entry Handle and character set of each active Entry will be returned. When the command can not be handled, NACK will be returned.

```
ListActiveEntries           ::= [APPLICATION tagListActiveEntries] SEQUENCE
{
                                COMPONENTS OF MsgHeader
}
```

Data transferred by TransferDataBlock command is as follows:

```
ActiveEntriesList           ::= SET OF ActiveEntries
```

```
ActiveEntries              ::= SEQUENCE
{
    groupHandle                [0] INTEGER,
    entryHandle                [1] INTEGER,
    charSetID                  [2] CharSetID
}
```

NACK Response

Name	Description	ReturnCode
rcNoActiveEntries	There is no active Entries.	178
rcCommandNotSupported	Command for Entry Operation is not supported.	218

4.2.5.9.GetEntryData command

Client	Server
GetEntryData =>	
	<= TransferDataBlock (EntryData)
ACK (NULL) =>	

This command is used to get an Active Entry data by specifying the Group by the Group handle and the Entry by the Entry handle. The Group handle and the Entry handle will be gotten by issuing ListActiveEntries command or GetActiveEntriesFieldData command. The returned data format and encoding for coded data and binary data will be specified by a client. When the command can not be handled, NACK will be returned.

```
GetEntryData               ::= [APPLICATION tagGetEntryData] SEQUENCE
{
                               COMPONENTS OF MsgHeader,
    groupHandle                [0] INTEGER,
    entryHandle                [1] INTEGER,
    exchangeDataFormat         [2] ExchangeDataFormat
}

ExchangeDataFormat         ::= CHOICE
{
    vCard                      [0] VCardEncoding
}

VCardEncoding              ::= SEQUENCE
{
    codedEncoding              [0] CodedEncoding,
    binaryEncoding             [1] BinaryEncoding
}
```

```
CodedEncoding                  ::= ENUMERATED
{
    sevenBit                       (0),
    base64Encoding                 (1),
    quotedPrintable                (2),
    eightBit                       (3)
}

BinaryEncoding                 ::= ENUMERATED
{
    base64Encoding                 (0),
    eightBit                       (1)
}
```

Data transferred by TransferDataBlock command is as follows:

```
EntryData                      ::= SEQUENCE
{
    charSetID                      [0] CharSetID,
    data                           [1] OCTET STRING
                                       --vCard format
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidGroupHandle	Specified Group Handle is invalid	141
rcInvalidEntryHandle	Specified Entry Handle is invalid	179
rcInvalidExchangeDataFormat	Specified exchange data format is invalid	168
rcExchangeDataFormatNotSupported	Specified exchange data format is not supported.	169
rcCodedEncodingNotSupported	Specified Coded encoding is not supported.	170
rcBinaryEncodingNotSupported	Specified Binary encoding is not supported.	171
rcCommandNotSupported	Command for Entry Operation is not supported.	218

4.2.5.10.GetActiveEntryData command

Client	**Server**

GetActiveEntryData =>	
	<= TransferDataBlock (ActiveEntryData)
ACK (NULL) =>	

This command is used to get an active Entry data by specifying the position in the active Entries. The position starts from one to show the first Entry in the active Entries. The returned data format and encoding for coded data and binary data will be specified by a client. When the command can not be handled, NACK will be returned.

```
GetActiveEntryData            ::= [APPLICATION tagGetActiveEntryData] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
    position                       [0] INTEGER,
    exchangeDataFormat             [1] ExchangeDataFormat
}

ExchangeDataFormat            ::= CHOICE
{
    vCard                          [0] VCardEncoding
}

VCardEncoding                 ::= SEQUENCE
{
    codedEncoding                  [0] CodedEncoding,
    binaryEncoding                 [1] BinaryEncoding
}

CodedEncoding                 ::= ENUMERATED
{
    sevenBit                       (0),
    base64Encoding                 (1),
    quotedPrintable                (2),
    eightBit                       (3)
}

BinaryEncoding                ::= ENUMERATED
{
    base64Encoding                 (0),
    eightBit                       (1)
}
```

Data transferred by TransferDataBlock command is as follows:

```
ActiveEntryData               ::= SEQUENCE
{
    groupHandle                    [0] INTEGER,
    entryHandle                    [1] INTEGER,
    charSetID                      [2] CharSetID,
    data                           [3] OCTET STRING
                                       -- vCard format
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidPosition	Specified position is invalid.	180
rcNoActiveEntries	There is no active Entries.	178
rcInvalidExchangeDataFormat	Specified exchange data format is invalid.	168
rcExchangeDataFormatNotSupported	Specified exchange data format is not supported.	169
rcCodedEncodingNotSupported	Specified Coded encoding is not supported.	170
rcBinaryEncodingNotSupported	Specified Binary encoding is not supported.	171
rcCommandNotSupported	Command for Entry Operation is not supported.	218

4.2.5.11.AddEntryData command

Client	Server

AddEntryData =>

<= ACK (EntryHandle)/NACK (ReturnCode)

This command is used to add an Entry data to the Group. The Group is specified by the Group Handle. An FU returns the Entry Handle assigned to the added Entry. This operation is permitted to a client who can access to the Group in write access mode. When the command can not be handled, NACK will be returned.

```
AddEntryData                ::= [APPLICATION tagAddEntryData] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   groupHandle                  [0] INTEGER,
   charSetID                    [1] CharSetID,
   dataFormat                   [2] DataFormat,
   data                         [3] OCTET STRING
                                    -- vCard format
}

DataFormat                  ::= ENUMERATED
{
   vCard                        (0)
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	EntryHandle	

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcNoRoomToAddReplace	The Group has no room to add an Entry data.	147
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidDataFormat	Specified data format is invalid or not supported.	167
rcInvalidReceivedDataFormat	Received data format is invalid.	181
rcCharacterSetNotSupported	Specified Character set is not supported	172
rcCommandNotSupported	Command for Entry Operation is not supported.	218

4.2.5.12.DeleteEntryData command

Client	Server

DeleteEntryData =>

<= ACK (NULL)/NACK (ReturnCode)

This command is used to delete an Entry in the Group. The Entry is specified by the Group Handle and the Entry Handle. This operation is permitted to a client who can access to the Group in write access mode. When the command can not be handled, NACK will be returned.

```
DeleteEntryData                 ::= [APPLICATION tagDeleteEntryData] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
   groupHandle                      [0] INTEGER,
   entryHandle                      [1] INTEGER
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidEntryHandle	Specified Entry Handle is invalid.	179
rcCommandNotSupported	Command for Entry Operation is not supported.	218

4.2.5.13.ReplaceEntryData command

Client	Server

ReplaceEntryData =>

<= ACK (NULL)/NACK (ReturnCode)

This command is used to replace the Entry data in the Group. The Entry to be replaced is specified by the Group Handle and the Entry Handle. This operation is permitted to a client who can access to the Group in write access mode. When the command can not be handled, NACK will be returned.

```
ReplaceEntryData            ::= [APPLICATION tagReplaceEntryData] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    groupHandle                 [0] INTEGER,
    entryHandle                 [1] INTEGER,
    charSetID                   [2] CharSetID,
    dataFormat                  [3] DataFormat,
    data                        [4] OCTET STRING
                                    -- vCard format
}

DataFormat                  ::= ENUMERATED
{
    vCard                       (0)
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcNoRoomToAddReplace	The Group has no room to replace an Entry data.	147
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidDataFormat	Specified data format is invalid or not supported.	167
rcInvalidReceivedDataFormat	Received data format is invalid.	181
rcCharacterSetNotSupported	Specified Character set is not supported	172
rcCommandNotSupported	Command for Entry Operation is not supported.	218
rcInvalidEntryHandle	Specified Entry Handle is invalid.	179

4.2.5.14.MoveEntryData command

Client	Server

MoveEntryData =>

<= **ACK (EntryHandle)/NACK (ReturnCode)**

This command is used to move an Entry from current Group to another Group. The Entry is specified by the Group Handle and the Entry Handle. To Group is specified by the Group Handle. An FU will return Entry Handle, which is assigned to the Entry with ACK. This operation is permitted to a client who can access to the 'from' and 'to' Groups in write access mode. When the command can not be handled, NACK will be returned.

```
MoveEntryData               ::= [APPLICATION tagMoveEntryData] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    fromGroupHandle             [0] INTEGER,
    fromEntryHandle             [1] INTEGER,
    toGroupHandle               [2] INTEGER
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	EntryHandle	

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcInvalidFromGroupHandle	Specified 'from' Group Handle is invalid.	142
rcInvalidToGroupHandle	Specified 'to' Group Handle is invalid.	143
rcInvalidEntryHandle	Specified Entry Handle is invalid.	179
rcCommandNotSupported	Command for Entry Operation is not supported.	218

4.2.5.15.CopyEntryData command

Client	**Server**

CopyEntryData =>

<= **ACK (EntryHandle)/NACK (ReturnCode)**

This command is used to copy Entry data into the Group. The Entry is specified by the Group Handle and the Entry Handle. To Group is specified by the Group Handle. An FU returns the Entry Handle with ACK which is assigned to the copied Entry. This operation is permitted to a client who can access to the Group in write access mode. When the command can not be handled, NACK will be returned.

```
CopyEntryData                ::= [APPLICATION tagCopyEntryData] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    fromGroupHandle              [0] INTEGER,
    fromEntryHandle              [1] INTEGER,
    toGroupHandle                [2] INTEGER
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	EntryHandle	

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcInvalidFromGroupHandle	Specified ‘from’ Group Handle is invalid.	142
rcInvalidToGroupHandle	Specified ‘to’ Group Handle is invalid.	143
rcInvalidEntryHandle	Specified Entry Handle is invalid.	179
rcInvalidGroupHandle	Specified to Group Handle is invalid.	141
rcCommandNotSupported	Command for Entry Operation is not supported.	218

4.2.5.16.SearchFieldData command

Client	Server

SearchFieldData =>

<= **ACK (SearchHandle, NumberOfActiveEntries)**
/NACK (ReturnCode)

This command is used to search the specific Field data in ALL Entries or CURRENT Active Entries which data are encoded by the specified character set in the specified Groups. If Search Handle value is zero, it shows that search operation will be performed to ALL Entries in the specified Groups. If specified search Handle is the returned value of the previous search operation, the search operation will be performed to the CURRENT Active Entries. (Note that even if ALL Entries or CURRENT active Entries are searched to find specific data, the Entries which data is encoded by the specified character set are the target Entries.) Field Name and its parameters are used to specify the Field. If Field name is ‘ALL’, all Fields will be searched. When an FU finds the Field data which meets the search condition, the Entry is marked as an Active Entry, then next search operation will be continued. When all Entries or all Active Entries are searched, an FU returns Search Handle and the numbers of Active Entries. When the command can not be handled, NACK is returned.

```
SearchFieldData              ::= [APPLICATION tagSearchFieldData] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    searchHandle                 [0] INTEGER,
    charSetID                    [1] CharSetID,
                                     -- to specify the Entry which data is encoded by this character set
    codedEncoding                [2] CodedEncoding,
                                     -- Value to be compare is encoded by specified encoding.
    searchCondition              [3] SearchCondition,
    groupHandleList              [4] GroupHandleList          OPTIONAL
                                     -- Optional for search operation to current Active Entries
}
```

```
NumberOfActiveEntries       ::= INTEGER

CodedEncoding               ::= ENUMERATED
{
    sevenBit                    (0),
    base64Encoding              (1),
    quotedPrintable             (2),
    eightBit                    (3)
}

SearchCondition             ::= CHOICE
{
    simpleFieldCompare          [0] SimpleFieldCompare,
    compoundCompare             [1] CompoundCompare
}

SimpleFieldCompare          ::= SEQUENCE
{
    howToCompare                [0] HowToCompare,
    fieldName                   [1] DisplayString,
                                    -- Field Name (Parameter, Parameter,..)
                                    -- Encoded by 8859-1 (US ASCII) character set
    value                       [2] DisplayString
                                    -- Only String data can be searched.
                                    -- Encoded by the specified encoding and character set
}

HowToCompare                ::= ENUMERATED
{
    equal                       (0),
    notEqual                    (1),
    greaterThan                 (2),
    greaterThanOrEqualTo        (3),
    lessThan                    (4),
    lessThanOrEqualTo           (5)
}

CompoundCompare             ::= SEQUENCE
{
    operand1                    [0] SearchCondition,
    connection                  [1] Connection,
    operand2                    [2] SearchCondition
}

Connection                  ::= ENUMERATED
{
    andConnect                  (0),
    orConnect                   (1)
}
```

GroupHandleList ::= SET OF GroupHandle

GroupHandle ::= INTEGER

ACK Response

Parameter Name	Data Type	Note
parameter-1	SearchHandle	
parameter-2	NumberOfActiveEntries	

NACK Response

Name	Description	ReturnCode
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidSearchHandle	Specified Search Handle is invalid.	182
rcInvalidCharacterSet	Specified Character set is not same as current active Entries.	173
rcDataNotFound	Data not found.	183
rcCodedEncodingNotSupported	Specified Coded encoding is not supported	170
rcCharacterSetNotSupported	Specified Character set is not supported	172
rcCommandNotSupported	Command for Entry/Field Operation is not supported.	218

4.2.5.17.GetActiveEntriesFieldData command

Client	**Server**

GetActiveEntriesFieldData =>
<= TransferDataBlock(ActiveEntriesFieldDataList)
ACK (NULL) =>

This command is used to get specified Field data of the current Active Entries. The Field is specified by the Field name and its parameter(s) and only string data Field can be specified. If an Entry has multiple Field data specified by the Field and/or parameter(s), all data will be returned separately with the Group handle and the Entry handle. The returned data encoding will be specified by a client. If an Entry does not have specified Field data, null data will be returned with the Group handle and the Entry handle. The Field data can be sorted before they are returned to a client. When the command can not be handled, NACK will be returned.

```
GetActiveEntriesFieldData      ::= [APPLICATION tagGetActiveEntriesFieldData] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
    fieldName                      [0] DisplayString,
                                       -- Field Name (Parameter, Parameter,..)
                                       -- Encoded by 8859-1 (US ASCII) character set
    codedEncoding                  [1] CodedEncoding,
    sort                           [2] Sort                OPTIONAL
}

CodedEncoding                  ::= ENUMERATED
{
    sevenBit                       (0),
    base64Encoding                 (1),
    quotedPrintable                (2),
    eightBit                       (3)
}

Sort                           ::= ENUMERATED
{
    ascendingBitOrder              (0),
    descendingBitOrder             (1),
    weightingFactor                (2),
    others                         (127)
}
```

Data transferred by TransferDataBlock command is as follows:

```
ActiveEntriesFieldDataList     ::= SET OF ActiveEntriesFieldData

ActiveEntriesFieldData         ::= SEQUENCE
{
    groupHandle                    [0] INTEGER,
    entryHandle                    [1] INTEGER,
    charSetID                      [2] CharSetID,
    value                          [3] DisplayString
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidFieldName	Specified Field name is invalid.	208
rcFieldDataNotFound	Specified Field data not found.	209
rcCodedEncodingNotSupported	Specified Coded encoding is not supported	170
rcSortNoSupport	Sort operation is not supported.	210
rcNoActiveEntries	There is no active Entries.	178
rcCommandNotSupported	Command for Entry/Field Operation is not supported.	218

4.3.[Schedule] Functional Unit

4.3.1.Overview

[Schedule] Functional Unit was dropped from this version of the specification. It will be supported in the later version.

Appendices

5.ASN.1 Tag

Salutation Packets are defined with Abstract Syntax Notation One (ASN.1) as defined by ISO 8824. In other words, a format of records carried in Salutation Packets is specified with a tag, which is specified with a **class** and a **number** within the class. There are four classes available in ASN.1, among which *Application* and *Context-Specific* tags are used to define tags in this specification.

A *Context-Specific* class is used to represent a tag for a field or record (data type) that is used in only a single or local context, such as in a parameter of a specific message.

Numbers for *Context-Specific* classes are defined when they are used.

An *Application* class is used to represent a tag for the messages defined for Functional Units under Salutation Personality Protocol. Numbers for *Application* classes are defined in the following sections

5.1.Common

```
-- ACK/NACK Response
tagACK                        INTEGER ::= 0
tagNACK                       INTEGER ::= 1

-- Data Transfer Messages
tagRequestDataTransfer        INTEGER ::= 100
tagDataBlockDescription       INTEGER ::= 101
tagTransferDataBlock          INTEGER ::= 102
tagRequestNextData            INTEGER ::= 103

-- Attribute Repository Messages
tagGetPrivateAttribute        INTEGER ::= 200
tagGetGlobalAttribute         INTEGER ::= 201
tagSetPrivateAttribute        INTEGER ::= 202
```

```
-- Job Related Messages
tagQueryJobStatus               INTEGER ::= 300
tagQueryJobEntryStatus          INTEGER ::= 301
tagNotifyJobStatus              INTEGER ::= 302
tagNotifyJobEntryStatus         INTEGER ::= 303
tagChangeJobAttribute           INTEGER ::= 304
tagChangeJobEntryAttribute      INTEGER ::= 305
tagSuspendJob                   INTEGER ::= 306
tagSuspendJobEntry              INTEGER ::= 307
tagResumeJob                    INTEGER ::= 308
tagResumeJobEntry               INTEGER ::= 309
tagCancelJob                    INTEGER ::= 310
tagCancelJobEntry               INTEGER ::= 311
tagFreeJobHandle                INTEGER ::= 312
tagStartMonitorJobStatus        INTEGER ::= 313
tagCancelMonitorJobStatus       INTEGER ::= 314

-- Dynamic Status Messages
tagQueryDynamicStatus           INTEGER ::= 400
tagSubscribeEvent               INTEGER ::= 401
tagNotifyEvent                  INTEGER ::= 402
tagUnsubscribeEvent             INTEGER ::= 403

-- Vendor Escape Messages
tagVendorEscape                 INTEGER ::= 999
```

5.2.Document Systems

```
-- [Print] Functional Unit
tagPrint                        INTEGER ::= 10000
tagListPrintJob                 INTEGER ::= 10001

-- [DOC Storage] Functional Unit
tagRetrieveDoc                  INTEGER ::= 11000
tagStoreDoc                     INTEGER ::= 11001
tagListFolderDoc                INTEGER ::= 11002
tagListFolder                   INTEGER ::= 11003
tagDeleteDoc                    INTEGER ::= 11004
tagChangeDocDesc                INTEGER ::= 11005
tagCopyDoc                      INTEGER ::= 11006
tagMoveDoc                      INTEGER ::= 11007
tagCreateFolder                 INTEGER ::= 11008
tagChangeFolderDesc             INTEGER ::= 11009
tagDeleteFolder                 INTEGER ::= 11010
```

```
-- [FAX Data Send] Functional Unit
tagSendFAX                    INTEGER ::= 12000
tagListFaxJob                 INTEGER ::= 12001
```

5.3.Voice Message Systems

```
--[Voice Message Storage] Functional Unit
tagListFolderContentVM        INTEGER ::= 20000
tagSendVM                     INTEGER ::= 20001
tagPlayVM                     INTEGER ::= 20002
tagSynthesizeVM               INTEGER ::= 20003
tagListVMSJob                 INTEGER ::= 20004
```

5.4.Personal Information Systems

```
-- [Address Book] Functional Unit
tagListGroups                 INTEGER ::= 30000
tagOpenGroup                  INTEGER ::= 30001
tagCloseGroup                 INTEGER ::= 30002
tagCreateGroup                INTEGER ::= 30003
tagDeleteGroup                INTEGER ::= 30004
tagRenameGroup                INTEGER ::= 30005
tagGetGroupData               INTEGER ::= 30006
tagListActiveEntries          INTEGER ::= 30007
tagGetEntryData               INTEGER ::= 30008
tagGetActiveEntryData         INTEGER ::= 30009
tagAddEntryData               INTEGER ::= 30010
tagDeleteEntryData            INTEGER ::= 30011
tagReplaceEntryData           INTEGER ::= 30012
tagMoveEntryData              INTEGER ::= 30013
tagCopyEntryData              INTEGER ::= 30014
tagSearchFieldData            INTEGER ::= 30015
tagGetActiveEntriesFieldData  INTEGER ::= 30016
```

6.Data Type Definition

Throughout the document, the data type for all the textual information is defined as "DisplayString".

```
DisplayString                    ::= OCTET STRING          -- Textual information
```

6.1.Service Description Record

In the following ASN.1 definitions of records, some elements of SEQUENCE types are flagged as OPTIONAL. However, depending on the context where the types are used, the elements are obligatory. The comment to the elements indicates in which contexts the elements must exist:

```
ServiceDescriptionRecord         ::= SEQUENCE
{
    functionalUnits                  [0] SET OF FunctionalUnitDescriptionRecord
}

FunctionalUnitDescriptionRecord  ::= SEQUENCE
{
    functionalUnitId                 [0] FunctionalUnitID,
    functionalUnitHandle             [1] FunctionalUnitHandle,
                                         -- Ignored in registered and requested FUDR
    attributes                       [2] SET OF AttributeRecord
}

FunctionalUnitID                 ::= INTEGER

FunctionalUnitHandle             ::= INTEGER

AttributeRecord                  ::= SEQUENCE
{
    attributeId                      [0] AttributeID,
                                     CHOICE
    {
        compareFunctionId            [1] CompareFunctionID,
                                         -- Used in registered or requested FUDR
        compareResult                [2] CompareResultCode
                                         -- Used in reply FUDR
    },
    value                            [3] ANY
}

AttributeID                      ::= INTEGER
```

```
CompareFunctionID              ::= ENUMERATED
{

-- Left     -hand operand = attribute of Registered Functional Unit Description Record
-- Right    -hand operand = attribute of Requested Functional Unit Description Record

     unspecified                    (0),      -- used in Query Capability
     --- for INTEGER (including ENUMERATED) types ---
     intLessThan                    (1),
     intLessThanOrEqualTo           (2),
     intGreaterThan                 (3),
     intGreaterThanOrEqualTo        (4),
     intEqualTo                     (5),
     intNotEqualTo                  (6),
     --- for BOOLEAN types ---
     boolEqualTo                    (11),
     boolNotEqualTo                 (12),
     --- for OCTET STRING (including DisplayString) types---
     strLessThan                    (21),
     strLessThanOrEqualTo           (22),
     strGreaterThan                 (23),
     strGreaterThanOrEqualTo        (24),
     strEqualTo                     (25),
     strNotEqualTo                  (26),
     strDoesContain                 (27),   -- contain as a substring
     strIsContained                 (28),   -- contained as a substring
     --- for SET OF INTEGER types ---
     setIntDoesContain              (61),
     setIntIsContained              (62),
     setIntEqualTo                  (63),
     setIntNotEqualTo               (64),
     setIntIntersect                (65),   -- at least one member is common
     --- for SET OF OCTET STRING types ---
     setStrDoesContain              (71),
     setStrIsContained              (72),
     setStrEqualTo                  (73),
     setStrNotEqualTo               (74),
     setStrIntersect                (75)    -- at least one member is common
}
```

```
CompareResultCode                   ::= ENUMERATED
{
    --- Comparison was performed ---
    true                                (0),
    --- Comparison was not performed ---
    attributeNotRegistered              (2),
    compareFunctionNotDefined           (3),
    wrongDataType                       (4),
    attributeNotRequested               (5)
}
```

6.2.Salutation Personality Protocol

Data types are listed in alphabetical order in each subsection.

6.2.1.Common

```
AbsoluteFunctionalUnitHandle        ::= SEQUENCE
{
    functionalUnit                      [0] FunctionalUnitHandle,
    nodeAddress                         [1] CHOICE
    {
        slmId                           [1] SLMID
    }                                       OPTIONAL  -- if omitted, the FU is registered with the same
                                                      -- SLM as the FU receiving this parameter
}

AttributeList                       ::= SET OF SEQUENCE
{
    attributeId                         [0] AttributeID,
    attributeValue                      [1] ANY       -- Type is defined by each attribute.
}

AuthenticationFlavor                ::= ENUMERATED
{
    null                                (0),
    userIdAndPassword                   (1)
}

CharSetID                           ::= INTEGER       -- Coded character set ID as registered with IANA
                                                      -- (MIB enumeration value is used)

DataHandle                          ::= INTEGER
```

```
DataLocation               ::= CHOICE
{
    client                     [0] NULL,                          -- default
    exportPool                 [1] NULL,                          -- for destination
    functionalUnit             [2] AbsoluteFunctionalUnitHandle,  -- for source
    url                        [3] DisplayString
                                   -- file system designated by URL of either ftp or file scheme
}

DataLocationScheme         ::= ENUMERATED
{
    salutation                 (0),   -- client/exportPool/functionalUnit allowed
    urlOfFtpScheme             (1),   -- URL with FTP scheme allowed
    urlOfFileScheme            (2)    -- URL with FILE scheme allowed
}

DataTransferMode           ::= ENUMERATED
{
    immediate                  (0),
    delayed                    (1)
}

DynamicStatusID            ::= INTEGER

FreeStorageSize            ::= INTEGER

JobEntriesStatus           ::= SET OF SEQUENCE
{
    jobEntryId                 [0] JobEntryID,
    jobEntryStatusCode         [1] JobStatusCode,
    jobEntryReasonCode         [2] ReasonCode           OPTIONAL
                                   -- present only if jobEntryStatusCode=suspended or error
}

JobDescription             ::= SEQUENCE
{
    jobHandle                  [0] JobHandle,
    requesterUserId            [1] UserID,
                                   -- "UserID" is set from the "UserID" specified in the Open Service
                                   -- request that has established a service session. Therefore, a client
                                   -- application must have registered as a [Client] FU to actually
                                   -- specify its "User ID" value so that it appears in the JobList.
    jobStatusCode              [2] JobStatusCode,
    dataSize                   [3] INTEGER              OPTIONAL,
    numOfJobEntries            [4] INTEGER              OPTIONAL
}

JobEntryID                 ::= INTEGER
```

```
JobHandle                       ::= INTEGER

JobList                         ::= SET OF JobDescription

JobPriority                     ::= INTEGER (0..100)        -- 50 is the normal priority.
                                                            -- 100 is the highest priority.

JobStatusCode                   ::= ENUMERATED
{
    completed                       (0),    -- job (entry) execution is successfully completed
    queued                          (1),    -- job (entry) execution has not begun yet
        -- "queued" is used in NotifyJob(Entry)Status only when “WaitingForScheduledTime” job (Entry) is
        -- queued
    started                         (2),    -- job (entry) execution has started and is being executed
    suspended                       (3),    -- job (entry) execution is suspended temporarily
    resumed                         (4),    -- suspended job (entry) execution is resumed
        -- "resumed" is never used in ACK response to QueryJob(Entry)Status
    error                           (5),    -- job (entry) execution is failed and was aborted
    waitingForScheduledTime         (6),    -- job (entry) execution is scheduled at a later specific time
        -- "waitingForScheduledTime" is never used in NotifyJob(Entry)Status
    canceled                        (7),    -- job (entry) was canceled by client’s request
    aborted                         (8)     -- job (entry) execution was aborted by client’s request
}

JobStatusNotificationMode       ::= SEQUENCE
{
    notificationTypeList            [0] SET OF JobStatusCode,
        -- The list of job or job entry status types to be notified
    requestJobEntriesStatus         [1] BOOLEAN   DEFAULT FALSE
        -- If the job-request-type command has no job entry or if “notificationTypeList” is empty,
        -- this parameter shall be omitted.
        -- Otherwise,
        --   When it is set to TRUE in the job-request-type command, “notificationTypeList” applies to
        --   all the job entries. Notification of command-level is not performed, but only notification of
        --   each job entry status is performed.
        --   When it is set to FALSE or omitted, notification of each job entry is not performed, but only
        --   command-level notification is performed.
}

Life                            ::= ENUMERATED
{
    job                             (0),
    session                         (1),
    persistent                      (2)
}
```

```
NotificationScheme              ::= CHOICE
{
    message                         [0] SEQUENCE
    {
        fu                          [0] FunctionalUnitHandle,
        nodeAddress                 [1] CHOICE
        {
            slmId                   [1] SLMID
        }
    }
}

ReasonCode                      ::= ENUMERATED
{
-- Values in the range of 0..127 are reserved for reason codes that are common to any jobs.

-- Values in the range of 128..32767 are reserved for job-specific reason codes.
-- They are defined by each Functional Unit specification.

-- Values larger than 32767 are reserved for vendor-specific reason codes.
}
```

```
ReturnCode                        ::= ENUMERATED
{
-- Values in the range of 0..127 are reserved for reason codes that are common to any messages.
    rcSyntaxError                     (1),
        -- The message has no APPLICATION tag.
        -- The message length is incorrect.
        -- The message data type is not SEQUENCE.
        -- The msgSeqId has a tag.
        -- The msgSeqId length is incorrect.
        -- The msgSeqId data type is not INTEGER.
        -- A mandatory parameter is missing.
        -- A parameter has no context-specific tag.
        -- A parameter length is incorrect.
        -- A parameter data type is incorrect.
    rcInvalidMessageHeader            (2),
        -- The msgSeqId sign is incorrect.
        -- The msgSeqId value is incorrect. (The message sender is attempting to initiate a new message
        --     sequence while another message sequence initiated by the sender is still active.)
    rcSystemError                     (3),
        -- Message execution failed due to a system error such as memory allocation error.
    rcTemporaryBusy                   (4),
        -- The message cannot be executed at the moment due to a resource shortage.
    rcUnsupportedMessage              (5),
        -- The message is unknown or not supported.
    rcUnauthorizedMessage             (6),
        -- The message is not supported for the current class of user.
    rcInvalidMessageSequence          (7),
    -- The message is not allowed in the current context of message sequence.
    rcUnsupportedParameter            (8)
    -- A parameter tag number is unknown or not supported.

-- Values in the range of 128..32767 are reserved for message-specific return codes.
-- They are defined by each Functional Unit specification.

-- Values larger than 32767 are reserved for vendor-specific return codes.
}

SimpleJobPriority                 ::= ENUMERATED
{
    low                               (0),
    normal                            (50),
    high                              (100)
}

SLMID                             ::= OCTET STRING (SIZE(16))
```

```
SpoolStorage                ::= BOOLEAN

SubscriptionHandle          ::= INTEGER

SupportedCommand            ::= ENUMERATED
{
-- Common command Optional Group
    vendorEscape                (0),
        --Support of vendorEscape (VendorEscape)
    notifyEventRelated          (1),
        -- Support of notifyEventRelated Optional Group (SubscribeEvent, UnsubscribeEvent/NotifyEvent)
    notifyJobStatusRelated      (2),
        -- Support of notifyJobStatusRelated Group (NotifyJobStatus/StartMonitorJobStatus/
        -- EndMonitorJobStatus/NotifyJobEntryStatus[5])
    jobEntryRelated             (3),
        -- Support of jobEntryRelated Optional Group (CancelJobEntry, ChangeJobEntryAttribute/
        -- QueryJobEntryStatus/NotifyJobEntryStatus/SuspendJobEntry/ResumeJobEntry 5)
-- Document Systems Optional Group
-- [Print] FU
        -- Reserved             (20),
-- [DOC Storage] FU
    folderHandlingRelated       (30),
        -- Support of folderHandlingRelated Optional Group (DeleteDoc/CopyDoc/MoveDoc/
        -- ChangeDocDesc/CreateFolder/DeleteFolder/ChangeFolderDesc)
-- [FAX Data Send] FU
        -- Reserved             (40),
-- [FAX Data] FU
    faxDocIDHandlingRelated     (50),
        -- Support of faxDocIdHandlingRelated Optional Group (RetrieveFaxDocID)
-- Personal Information Systems Optional Group
-- [Address Book] FU
    entryOperationRelated       (70),
        -- Support of entryOperationRelated Optional Group (CreateGroup/DeleteGroup/RenameGroup/
        -- ListActiveEntries/GetEntryData/GetActiveEntryData/AddEntryData/DeleteEntryData/
        -- ReplaceEntryData/MoveEntryData/CopyEntryData/SearchFieldData/GetActiveEntriesFieldData)
-- Voice Message Systems Optional Group
-- [Voice Message Storage] FU
    synthesizeOpeRelated        (90)
        -- Support of synthesizeOpeRelated Optional Group (SynthesizeVM)
}

TelephoneNumberString       ::= DisplayString
                                    -- FROM(    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" |
                                    -- " " | "(" | ")" | "-" | "+" | "," | "*" | "#" |
                                    -- "A" | "B" | "C" | "D" )
```

[5] NotifyJobEntryStatus command is supported when both of "notifyJobStatusRelated" and "jobEntryRelated" are set.

```
UserID                          ::= DisplayString
```

6.2.2.Document Systems

```
ByteFillOrder                   ::= ENUMERATED
{
-- Following value is meaningful when document data format is biLevelImageStream or tiff.
-- ByteFillOrder shows the bit order in the image data byte.
-- When image data is raw data (not compressed), it shows the Byte Fill Order of raw image
-- data. When image data is compressed, it shows the Byte Fill Order of compressed data.

--      addr0   addr1   addr2   addr3
--      [0..7]  [8..15] [16..23][24..31] ..  ByteFillOrder=msb case
--      [7..0]  [15..8] [23..16][31..24] ..  ByteFillOrder=lsb case

    msb                             (0),
    lsb                             (1)
}
```

```
DataFormat                        ::= ENUMERATED
{
    notSpecified                       (127),

--Document Related Data Format Start
--
-- Image Bitstream listed bellow. When selected, ImageStreamAttributes are referred to
    biLevelImageStream                 (1000),          --Three ImageSize types are supported
                                                        -- When this data format is set, the image size
                                                        -- attribute can be selected from "axisSize",
                                                        -- "totalSize" or "pageDimensions".
    biLevelImageStreamAxisSize         (1001),          -- axisSize in ImageSize is supported.
                                                        -- When this data format is set, the image size
                                                        -- attribute must be "axisSize".
    biLevelImageStreamTotalSize        (1002),          -- totalSize in ImageSize is supported.
                                                        -- When this data format is set, the image size
                                                        -- attribute must be "totalSize".
    biLevelImageStreamPageDimension (1003),             -- pageDimension in imageSize is supported.
                                                        -- When this data format is set, the image size
                                                        -- attribute must    be "pageDimensions".

-- Structured Image Data listed bellow
    ms53A12                            (1010),
    tiff                               (1011),
    bmp                                (1012),
    pcx                                (1013),
    dcx                                (1014),
    winMetaFile                        (1015),
    os2MetaFile                        (1016),
    xdw                                (1017),          -- DocuWorks image format. Fuji Xerox Co. Ltd.
    jfif                               (1018),          -- Color image format

-- Printer Datastream listed bellow.
-- Each PDL needs the version information. PDL    version will be further studied.
    langPCL                            (1203),          -- Printer Control Language. Hewlett-Packard Co.
    lang201PL                          (1204),          -- NEC Co.
    langPJL                            (1205),          -- Printer Job Language. Hewlett-Packard Co.
    langPS                             (1206),          -- PostScript(TM). Post Script is a trademark of
                                                        -- Adobe Systems Inc.
    langEscapeP                        (1209),          -- EPSON ESC/P(TM). Epson Co.
    langLIPS                           (1239),          -- LBP Image Processing System. Canon Inc.
    langIPDS                           (1250),          -- Intelligent Printer Data Stream,
                                                        -- IBM Printing Systems. Corresponds to
                                                        -- langIPDS(7) of RFC1759.
    langPAGES                          (1251),          -- Page Printer Advanced Graphics Escape Set.
                                                        -- IBM Japan Ltd.
    langMODCA                          (1252),          -- Mixed Object Document Content Architecture,
                                                        -- IBM Printing Systems. Corresponds to
                                                        -- langIPDS(15) of RFC1759.
    langRPDL                           (1260),          -- Ricoh    Corp.
    langART                         (1270),         -- Fuji Xerox Co. Ltd.

-- Unstructured Text Data listed bellow. (for further study)
    plainText                       (1400),
--
-- Structured Text Data (for further study)
--
-- Portable Document
    pdf                             (1600)          -- Portable Document Format,
                                                    -- Adobe Systems Inc.
-- Other Types (for further study)
--
-- Document Related Data Format End
}

DocFormatInterpretation         ::= CHOICE
{
    imageStreamAttributes           [0] ImageStreamAttributes
                                    -- Chosen when documentDataFormat is
                                    -- biLevelImageStream
--
-- Other interpretation attributes are for further study
}

DocumentDataDescriptor          ::= SEQUENCE
{
    documentDataFormat              [0] DataFormat,
    documentFormatInterpretation    [1] DocFormatInterpretation         OPTIONAL
}

ImageCompAlgorithm              ::= ENUMERATED
-- Following value is meaningful when document data format is biLevelImageStream or tiff.

{
    raw                             (0),
    mh                              (1),
    mhb                             (2),            -- EOL Byte Boundary
    mr                              (3),
    mrb                             (4),            -- EOL Byte Boundary
    mmr                             (5),
    jpeg                            (6),            -- Compression for color image
    jbig                            (7),            -- Progressive Bi-level Image Compression
                                                    -- ITU-T Recommendation T.82
    other                           (127)
}
```

```
ImageResolution               ::= ENUMERATED
-- Following value is meaningful when document data format is biLevelImageStream , tiff, bmp, pcx
-- or dcx.

{
   normal                         (0),           -- 8x3.85l/mm
   fine                           (1),           -- 8x7.7l/mm
   semi-superFine                 (2),           -- 8x15.4l/mm
   superFine                      (3),           -- 16x15.4l/mm
   dpi180                         (4),           -- 180dpi
   dpi200                         (5),           -- 200dpi
   dpi240                         (6),           -- 240dpi
   dpi300                         (7),           -- 300dpi
   dpi360                         (8),           -- 360dpi
   dpi400                         (9),           -- 400dpi
   dpi600                         (10),          -- 600dpi
   dpi720                         (11),          -- 720dpi
   dpi800                         (12),          -- 800dpi
   dpi1200                        (13),          -- 1200dpi
   dpi200x100                     (30),          -- 200x100dpi G4 Optional
   dpi100                         (31)           -- 100dpi
}

ImageSize::= CHOICE
{
   axisSize                       [0] SEQUENCE
   {
      xAxisSize                   [0] INTEGER,   -- Unit : dot
      yAxisSize                   [1] INTEGER    -- Unit : dot
   },
   totalSize                      [1] INTEGER,   -- Unit : byte
   pageDimensions                 [2] PageDimensions
}

ImageStreamAttributes         ::= SEQUENCE
{
                                     -- All parameters shall be specified for "biLevelImageStream"
                                     -- document format.
   imageSize                      [0] ImageSize,
   imageCompAlgorithm             [1] ImageCompAlgorithm,
   imageByteFillOrder             [2] ByteFillOrder,
   imageResolution                [3] ImageResolution
}
```

```
MaximumRecordingLength          ::= ENUMERATED
{
    a4                              (0),
    b4                              (1),
    unlimited                       (2)
}

PageDimensions                  ::= SEQUENCE
{
    recordingWidth                  [0] RecordingWidth,
    maximumRecordingLength          [1] MaximumRecordingLength
}

RecordingWidth                  ::= ENUMERATED
{
    rw864                           (0),
    rw1216                          (1),
    rw1728                          (2),
    rw2048                          (3),
    rw2432                          (4)
}
```

6.2.2.1.[Print] Functional Unit

```
ExcerptPrintJobDescription      ::= SEQUENCE
{
    jobHandle                       [0] JobHandle,
    jobStatusCode                   [1] JobStatusCode,
    printPriority                   [2] SimpleJobPriority
}
```

```
ListExcerptPrintJob            ::= SET OF ExcerptPrintJobDescription

PaperDirection                 ::= ENUMERATED
{
   portrait                       (1),
   landscape                      (2),
   other                          (127)
}

PaperSize                      ::= ENUMERATED
{
                                        -- reference from printer MIB's recommendation.
   letter                         (1),  -- North American letter size: 8.5 by 11 inches
   legal                          (2),  -- North American legal size:  8.5 by 14 inches
   na-10x13-envelope              (3),  -- North American 10x13 envelope size:  10 by 13 inches
   na-9x12-envelope               (4),  -- North American 9x12 envelope size:  9 by 12 inches
   na-number-10-envelope          (5),  -- North American number 10 business envelope
                                        -- size:  4.125 by 9.5 inches
   na-7x9-envelope                (6),  -- North American 7x9 size:  7 by 9 inches
   na-9x11-envelope               (7),  -- North American 9x11 size: 9 by 11 inches
   na-10x14-envelope              (8),  -- North American 10x14 envelope size: 10 by 14 inches
   na-number-9-envelope           (9),  -- North American number 9 business envelope
   na-6x9-envelope                (10), -- North American 6x9 envelope size:  6 by 9 inches
   na-10x15-envelope              (11), -- North American 10x15 envelope size: 10 by 15 inches
   a                              (12), -- engineering A size 8.5 inches by 11 inches
   b                              (13), -- engineering B size 11 inches by 17 inches
   c                              (14), -- engineering C size 17 inches by 22 inches
   d                              (15), -- engineering D size 22 inches by 34 inches
   e                              (16), -- engineering E size 34 inches by 44 inches
   iso-a0                         (17), -- ISO A0  size:  841 mm by 1189 mm
   iso-a1                         (18), -- ISO A1  size:  594 mm by  841 mm
   iso-a2                         (19), -- ISO A2  size:  420 mm by  594 mm
   iso-a3                         (20), -- ISO A3  size:  297 mm by  420 mm
   iso-a4                         (21), -- ISO A4  size:  210 mm by  297 mm
   iso-a5                         (22), -- ISO A5  size:  148 mm by  210 mm
   iso-a6                         (23), -- ISO A6  size:  105 mm by  148 mm
   iso-a7                         (24), -- ISO A7  size:   74 mm by  105 mm
   iso-a8                         (25), -- ISO A8  size:   52 mm by   74 mm
   iso-a9                         (26), -- ISO A9  size:   37 mm by   52 mm
   iso-a10                        (27), -- ISO A10 size:   26 mm by   37 mm
   iso-b0                         (28), -- ISO B0  size: 1000 mm by 1414 mm
   iso-b1                         (29), -- ISO B1  size:  707 mm by 1000 mm
   iso-b2                         (30), -- ISO B2  size:  500 mm by  707 mm
   iso-b3                         (31), -- ISO B3  size:  353 mm by  500 mm
   iso-b4                         (32), -- ISO B4  size:  250 mm by  353 mm
   iso-b5                         (33), -- ISO B5  size:  176 mm by  250 mm
   iso-b6                         (34), -- ISO B6  size:  125 mm by  176 mm
   iso-b7                         (35), -- ISO B7  size:   88 mm by  125 mm
   iso-b8                         (36), -- ISO B8  size:   62 mm by   88 mm
```

```
    iso-b9                  (37),  -- ISO B9   size:     44 mm by     62 mm
    iso-b10                 (38),  -- ISO B10 size:     31 mm by     44 mm
    iso-c0                  (39),  -- ISO C0 size:     917 mm by 1297 mm
    iso-c1                  (40),  -- ISO C1 size:     648 mm by   917 mm
    iso-c2                  (41),  -- ISO C2 size:     458 mm by   648 mm
    iso-c3                  (42),  -- ISO C3 size:     324 mm by   458 mm
    iso-c4                  (43),  -- ISO C4 size:     229 mm by   324 mm
    iso-c5                  (44),  -- ISO C5 size:     162 mm by   229 mm
    iso-c6                  (45),  -- ISO C6 size:     114 mm by   162 mm
    iso-c7                  (46),  -- ISO C7 size:      81 mm by   114 mm
    iso-c8                  (47),  -- ISO C8 size:      57 mm by    81 mm
    iso-designated          (48),  -- ISO Designated Long
                                   -- size:   110 mm by 220 mm
    jis-b0                  (49),  -- JIS B0   size   1030 mm by 1456 mm
    jis-b1                  (50),  -- JIS B1   size     728 mm by 1030 mm
    jis-b2                  (51),  -- JIS B2   size     515 mm by   728 mm
    jis-b3                  (52),  -- JIS B3   size     364 mm by   515 mm
    jis-b4                  (53),  -- JIS B4   size     257 mm by   364 mm
    jis-b5                  (54),  -- JIS B5   size     182 mm by   257 mm
    jis-b6                  (55),  -- JIS B6   size     128 mm by   182 mm
    jis-b7                  (56),  -- JIS B7   size      91 mm by   128 mm
    jis-b8                  (57),  -- JIS B8   size      64 mm by    91 mm
    jis-b9                  (58),  -- JIS B9   size      45 mm by    64 mm
    jis-b10                 (59),  -- JIS B10 size      32 mm by    45 mm
    na-8X13-legal           (70),  -- governmental legal   8 inches by 13 inches
    hagaki                  (71),  -- Hagaki size       100 mm by   148 mm
    half-letter             (72),  -- North American half letter size: 5.5 by 8.5 inches
    others                  (127)
}

PersonalityProtocol       ::= ENUMERATED
{
    salutationPrint         (1),
    other                   (127),

-- For Emulated Personality or Native Personality

    langPCL                 (1203),
    langPJL                 (1205),
    langPS                  (1206),
    win32RawPrtData         (1211),          -- Raw Print Data Stream for 32-bit Windows OS
    langLIPS                (1239)
}
```

```
PrintControlAttribute          ::= SEQUENCE
{
    printPaperSize                 [0] PaperSize                     OPTIONAL,
    printResolution                [1] ImageResolution               OPTIONAL,
    printPaperDirection            [2] PaperDirection                OPTIONAL,
    printCopyCount                 [3] INTEGER                       OPTIONAL,
    printPaperInputSelect          [4] PrintPaperInputSelect         OPTIONAL,
    printPaperOutputSelect         [5] PrintPaperOutputSelect        OPTIONAL,
    printOutputBinSelect           [6] PrintOutputBinSelect          OPTIONAL,
    printDuplexMode                [7] PrintDuplexMode               OPTIONAL,
    printFaceUpMode                [8] PrintFaceUpMode               OPTIONAL,
    printPriority                  [9] SimpleJobPriority             OPTIONAL,
    printStaplingSelect            [10] PrintStaplingSelect          OPTIONAL,
    printFileName                  [11] DisplayString                OPTIONAL
}

PrintDuplexMode                ::= SEQUENCE
{
    printDuplexModeSelect          [0] PrintDuplexModeSelect,
    bindingMargin                  [1] INTEGER    OPTIONAL           -- 0.1 mm
}

PrintDuplexModeSelect          ::= ENUMERATED
{
    simplex                        (0),
    left-binding-duplex            (1),
    right-binding-duplex           (2),
    top-binding-duplex             (3),
    other                          (127)
}

PrinterErrorDescription        ::= SEQUENCE
{
    printerStatusCode              [0] PrinterStatusCode,
    systemError                    [1] DisplayString,         -- detail description
    others                         [2] DisplayString-- detail description
}

PrinterErrorDetail             ::= SET OF PrinterErrorDescription
```

```
PrinterStatusCode               ::= ENUMERATED
{
    noPaper                         (0),
    noToner                         (1),
    doorOpen                        (2),
    jammed                          (3),
    offline                         (4),
    receiving                       (5),
    error                           (6),
    normal                          (7),
    paperNearEnd                    (8),
    tonerNearEnd                    (9),
    fatalError                      (10),                  -- errors requiring equipment repair
    others                          (127)
}

PrintFaceUpMode                 ::= ENUMERATED
{
    faceDown                        (1),
    faceUp                          (2),
    other                           (127)
}

PrinterOperationStatus          ::= SET OF PrinterStatusCode

PrinterPaperInputTray           ::= SET OF PrinterPaperInputTrayStatus

PrinterPaperInputTrayStatus     ::= SEQUENCE
{
    printPaperInputSelect           [0] PrintPaperInputSelect,
    paperSize                       [1] PaperSize,
    paperDirection                  [2] PaperDirection,
    paperExistence                  [3] BOOLEAN            OPTIONAL
                                        -- If TRUE, input tray is not empty
                                        -- If FALSE, input tray is empty
}
```

```
PrintJobDescription            ::= SEQUENCE
{
    jobHandle                      [0] JobHandle,
    requesterUserId                [1] UserID,
                                       -- "UserID" is set from the "UserID" specified in the Open
                                       -- Service request that has established a service session.
                                       -- Therefore, a client application must have registered as a
                                       -- [Client] FU to actually specify its "User ID" value so that it
                                       -- appears in the JobList.
    jobStatusCode                  [2] JobStatusCode,
    dataSize                       [3] INTEGER            OPTIONAL,
    queuedTime                     [4] DisplayStringOPTIONAL,
    printPriority                  [5] SimpleJobPriority  OPTIONAL,
    printCopyCount                 [6] INTEGER            OPTIONAL,
    printPageCount                 [7] INTEGER            OPTIONAL,
    printFileName                  [8] DisplayString      OPTIONAL
}

PrintJobList                   ::= SET OF PrintJobDescription

PrintOutputBinSelect           ::= INTEGER        -- equipment default bin is to be taken when Zero
                                                  -- (0) is specified.

PrintPaperOutputSelect         ::= ENUMERATED
{
    standard                       (0),
    collatedSort                   (1),
                                       -- corresponds to “putOutputPageCollated” of IETF RFC 1759.
                                       -- page collation. Thus each stack contains identical pages.
    stack                          (2),
    nonCollatedSort                (3),
                                       -- corresponds to “putOutputDecollating” of IETF RFC 1759.
                                       -- individual pages of multi-part form are separated and stored
                                       -- into separate stacks for each part. Thus a stack contains a set of
    other                          (127)
}

PrintPaperInputSelect          ::= ENUMERATED
{
    manualFeed                     (0),
    tray-1                         (1),
    tray-2                         (2),
    tray-3                         (3),
    tray-4                         (4),
    tray-5                         (5),
    automaticSelect                (126),
                                       -- [Print] FU selects an input tray by PaperSize.
    other                          (127)
}
```

```
PrintStaplingSelect          ::= ENUMERATED
{
    nonStaple                    (0),
    withStapleLeftCorner         (1),    -- UpperLeft Corner
    withStapleRightCorner        (2),    -- UpperRight Corner
    withStapleLeftSingle         (3),
    withStapleRightSingle        (4),
    withStapleTopSingle          (5),
    withStapleLeftDouble         (6),
    withStapleRightDouble        (7),
    withStapleTopDouble          (8),
    withStapleOthers             (127)
}
```

6.2.2.2.[FAX Data Send] Functional Unit

```
CoverSheetGen                ::= BOOLEAN

CSInfo                       ::= SEQUENCE
{
    jobEntryId                   [0] JobEntryID,
    faxNumber                    [1] TelephoneNumberString,
    subAddressNumber             [2] DisplayString          OPTIONAL,
    name                         [3] DisplayString          OPTIONAL,
    section                      [4] DisplayString          OPTIONAL,
    company                      [5] DisplayString          OPTIONAL,
    phoneNumber                  [6] TelephoneNumberString          OPTIONAL,
    address                      [7] DisplayString          OPTIONAL,
    faxProtocol                  [8] FAXProtocol                    DEFAULT g3,
    orderingData                 [9] TelephoneNumberString          OPTIONAL
                                     -- Ordering data is sent out by DTMF(Dual Tone Multi-
                                     -- Frequency, G3) prior to G3 communication or UUI
                                     -- (User User Information, G4) in G4 communication.
}

FaxControlAttribute          ::= SEQUENCE
{
    tsInfo                       [0] TSInfo                         OPTIONAL,
    csInfoGroup                  [1] SET OF CSInfo,
    requestPriority              [2] SimpleJobPriority              OPTIONAL,
    retryCount                   [3] INTEGER                        OPTIONAL
}

FaxJobList                   ::= SET OF FaxJobDescription
```

```
FaxJobDescription           ::= SEQUENCE
{
    jobHandle                   [0] JobHandle,
    requesterUserId             [1] UserID,
                                    -- "UserID" is set from the "UserID" specified in the Open Service
                                    -- request that has established a service session. Therefore, a client
                                    -- application must have registered as a [Client] FU to actually
                                    -- specify its "User ID" value so that it appears in the JobList.
    jobStatusCode               [2] JobStatusCode,
    dataSize                    [3] INTEGER             OPTIONAL,
    numOfJobEntries             [4] INTEGER             OPTIONAL
}

FAXProtocol                 ::= ENUMERATED      -- G3 is assumed when omitted.
{
    g3                          (1),
    g4                          (2),
    auto                        (3)             -- Automatic selection of FaxProtocol to be used
}

FaxSendErrorStatus          ::= SEQUENCE
{
    systemError                 [0] DisplayString,      -- detail description
    others                      [1] DisplayString-- detail description
}

FAXSendFreeStorageSize      ::= INTEGER

FaxSendOrdering             ::= BOOLEAN
                                    -- FaxSendOrdering is used in, for example, facsimile network
                                    -- and mail service. “phoneNumber” specifies a phone
                                    -- number for a service and orderingData is for final recipient
                                    -- number

FaxSendStatus               ::= ENUMERATED
{
    powerFailure                (0),
    warmingUp                   (1),
    offline                     (2),
    ready                       (3),
    sending                     (4),
    receiving                   (5),
    error                       (6),
    others                      (127)
}

NumOfCalledSubscribers      ::= INTEGER

PageHeaderGen               ::= BOOLEAN
```

```
PersonalityProtocol              ::= ENUMERATED
{
    salutationFaxDataSend            (1),
    other                            (127)
}

TSInfo                           ::= SEQUENCE
{
    name                             [0] DisplayString        OPTIONAL,
    section                          [1] DisplayString        OPTIONAL,
    company                          [2] DisplayString        OPTIONAL,
    phoneNumber                      [3] TelephoneNumberString             OPTIONAL,
    faxNumber                        [4] TelephoneNumberString             OPTIONAL,
    address                          [5] DisplayString        OPTIONAL,
    subject                              [6] DisplayString                 OPTIONAL,
    coverSheetGen                    [7] CoverSheetGen                     OPTIONAL,
    memoForCover                     [8] DisplayString        OPTIONAL,
    pageHeaderGen                    [9] PageHeaderGen                     OPTIONAL,
    memoForHeader                    [10] DisplayString                    OPTIONAL
}
```

-- The following are for Future Study, and not referred to in this release.

```
-- EventHistory                  ::= (for further study)

-- FAXDataSendStatus             ::= SET
-- {
--   serverStatus                    [0] ServerStatus         OPTIONAL,
--   numberOfExistingJob             [1] NumOfExistingJob     OPTIONAL,
--   numberOfAvailJobEntry           [2] NumOfAvailJobEntry   OPTIONAL
-- }

-- FAXInfoItem                   ::= SET
-- {
--   faxDataSendStatus               [0] FAXDataSendStatus    OPTIONAL,
--   jobList                             [1] JobList          OPTIONAL,
--   eventHistory                    [2] EventHistory         OPTIONAL
-- for further study
-- }

-- ScheduledAfter                ::= DisplayString    -- "hh:mm:ss" format

-- ScheduledDateTime             ::= DisplayString    -- "yy mm dd hh:mm:ss zzz" format
                                                      --         zzz = JST, GMT, etc.
-- ScheduledTime                ::= CHOICE
-- {
--                                 [0] ScheduledDateTime,
--                                 [1] ScheduledAfter
-- }

-- SelectFAXInfoItem            ::= ENUMERATED
-- {
--   faxDataSendStatus             (1),
--   jobList                       (2),
--   eventHistory                  (3)
-- for further study
-- }

-- ServerStatus                 ::= DisplayString           -- (further study for the detail)

-- TransmitSpeed                ::= ENUMERATED
-- {
--   s2400                         (1),            -- If not specified, the speed is determined
--   s4800                         (2),            -- via the negotiation on the FAX protocol.
--   s7200                         (3),
--   s9600                         (4),
--   s12000                        (5),
--   s14400                        (6)
--}
```

6.2.2.3.[DOC Storage] Functional Unit

```
AccessMode                      ::= ENUMERATED
{
   readOnly                        (1),
   readWrite                       (2),
   other                           (127)
}

DataContent                     ::= CHOICE
{
   documentData                    [0] NULL,                -- default (Content is document data)
   fileData                        [1] FileData             -- (Content is file data)
}

DataStoreMode                   ::= ENUMERATED
{
   documentDataMode                (1),
                                      -- Non-transparent Mode
   fileMode                        (2)
}

DocComment                      ::= DisplayString
```

```
DocDescription          ::= SEQUENCE
{
    documentId              [0] DocumentID,
    ownerName               [1] OwnerName               OPTIONAL,
    docComment              [2] DocComment              OPTIONAL,
    createDateTime          [3] DisplayString           OPTIONAL,
    size                    [4] INTEGER                 OPTIONAL,
                                -- size in bytes of this document
    numberOfBlocks          [5] INTEGER                 OPTIONAL,
                                -- size in blocks that may be useful in RetrieveDoc to specify
                                -- startDataBlock and endDataBlock parameter.
    typeOfContent           [6] DataContent             OPTIONAL
}

DocList                 ::= SET OF DocDescription

DocumentID              ::= INTEGER

FileData                ::= SEQUENCE
{
    fileName                [0] DisplayString,
    fileType                [1] FileType                OPTIONAL,
    fileTitle               [2] DisplayString           OPTIONAL,
    fileComment             [3] DisplayString           OPTIONAL,
    fileCreateDateTime      [4] DisplayString           OPTIONAL,
    fileSize                [5] INTEGER                 OPTIONAL
                                -- file size in bytes
}

FileType                ::= ENUMERATED
{
    deviceDriver            (0),                -- Device Driver
    applicationProgram      (1),                -- Application program
    executable              (2),                -- Executable code
    applicationData         (3),                -- Application data
    other                   (127)
}

FolderComment           ::= DisplayString
```

```
FolderDescription               ::= SEQUENCE
{
    folderId                        [0] FolderID,
    ownerName                       [1] OwnerName              OPTIONAL,
    folderComment                   [2] FolderComment          OPTIONAL,
    createDateTime                  [3] DisplayString          OPTIONAL,
    usedSize                        [4] INTEGER                OPTIONAL,
                                        -- size in bytes occupied by the documents in this folder
    freeSize                        [5] INTEGER                OPTIONAL,
                                        -- size in bytes of free area in this folder
    numberOfDocuments               [6] INTEGER                OPTIONAL
}

FolderID                        ::= INTEGER
                                    -- FolderID=0 is used for Default Public Folder.

FolderList                      ::= SET OF FolderDescription

OperatorIntervention            ::= SEQUENCE
{
    requiredAction                  [0] DisplayString
}

OperatorInformation             ::= SEQUENCE
{
    information                     [0] DisplayString
}

OwnerName                       ::= DisplayString

PersonalityProtocol             ::= ENUMERATED
{
    salutationDocStorage            (1),
    other                           (127)
}

StorageSize                     ::= INTEGER
```

6.2.3.Voice Message Systems

6.2.3.1.[Voice Message Storage] Functional Unit

```
DeliveryGrade                   ::= ENUMERATED
{
    urgent                          (0),
    normal                          (1),
    nonUrgent                       (2)
}
```

```
DescriptiveComment              ::= DisplayString

Encoding                        ::= SEQUENCE
{
    encodingAlgorithm               [0] EncodingAlgorithm,
    samplingRate                    [1] SamplingRate            OPTIONAL
}

EncodingAlgorithm               ::= ENUMERATED
{
    analog                          (0),
    pcm                             (1),
    u-law                           (2),
    a-law                           (3),
    adpcm                           (4),
    cvsd                            (5),
    apc-ab                          (6),
    ld-celp                         (7),
    v-celp                          (8),
    others                          (127)
}

FolderID                        ::= INTEGER
                                    -- FolderID=0 is used for Default Public Folder.

HeaderInformation               ::= BIT STRING
{
    senderId                        (0),
    dateSent                        (1)
}

VMSJobList                          ::= SET OF VMSJobDescription

VMSJobDescription               ::= SEQUENCE
{
    jobHandle                       [0] JobHandle,
    jobStatusCode                   [1] JobStatusCode,
    numOfJobEntries                 [2] INTEGER
}

PersonalityProtocol             ::= ENUMERATED
{
    SalutationVMS                   (1),
    others                          (127)
}
```

```
PlayVMStatus                ::= ENUMERATED
{
    playing                     (0),
    suspended                   (1),
    position                    (2),
    error                       (3),
    others                      (127)
}

Receiver                    ::= SEQUENCE
{
    jobEntryId                  [0] JobEntryID,
    receiverId                  [1] CHOICE
    {
        userId                  [0] UserID,
        telephoneNo             [1] TelephoneNumberString
    },
    deferredDeliveryTime        [2] UTCTime             OPTIONAL
                                    -- sender specifies the date and time up to which delivery
                                    -- of the message should be deferred for this receiver
}

Recipient                   ::= SEQUENCE
{
    jobEntryId                  [0] JobEntryID,
    recipientId                 [1] UserID,
    recipientType               [2] ENUMERATED
    {
        primary                 (0),
        copy                    (1),
        blindCopy               (2)
    },
    deferredDeliveryTime        [3] UTCTime             OPTIONAL
                                    -- sender specifies the date and time up to which delivery
                                    -- of the message should be deferred
}

SamplingRate                ::= ENUMERATED
{
    -- r4K                      (0),
    -- r8K                      (1),
    -- r16K                     (2),
    r24K                        (3),
    r32K                        (4)
    -- r64K                     (5),
    -- others                   (127)
}
```

```
TextLanguage                    ::= DisplayString
                                     -- Language tag which is defined in RFC 1766.
                                     -- Language tag consists of primary tag which is ISO 639
                                     -- language and secondary tag which is ISO 3166 country/area
                                     -- in which the language is used.

VoiceMessageDataDescriptor      ::= SEQUENCE
{
    voiceMessageDataFormat           [0] VoiceMessageDataFormat,
    voiceMessageFormatInterpretation     [1] VoiceMessageFormatInterpretation
}

VoiceMessageDataFormat          ::= ENUMERATED
{
    voiceMessage                     (0)
}

VoiceMessageDescriptor          ::= SEQUENCE
{
    voiceMsgId                       [0] VoiceMsgID,
    descriptiveComment               [1] DescriptiveComment    OPTIONAL
}

VoiceMessageFormatInterpretation    ::= CHOICE
{
    voiceMessageEncoding             [0] Encoding
}

VoiceMsgID                      ::= INTEGER

VoiceMsgList                    ::= SET OF VoiceMessageDescriptor

VoiceType                       ::= ENUMERATED
{
    maleVoicePreferred               (125),
    femaleVoicePreferred             (126),
    dontCare                         (127)
}
```

6.2.4.Personal Information Systems

6.2.4.1.[Address Book] Functional Unit

```
ActiveEntries                   ::= SEQUENCE
{
    groupHandle                      [0] INTEGER,
    entryHandle                      [1] INTEGER,
    charSetID                        [2] CharSetID
}
```

```
ActiveEntriesFieldData          ::= SEQUENCE
{
    groupHandle                     [0] INTEGER,
    entryHandle                     [1] INTEGER,
    charSetID                       [2] CharSetID,
    value                           [3] DisplayString
}

ActiveEntriesFieldDataList      ::= SET OF ActiveEntriesFieldData

ActiveEntriesList               ::= SET OF ActiveEntries

ActiveEntryData                 ::= SEQUENCE
{
    groupHandle                     [0] INTEGER,
    entryHandle                     [1] INTEGER,
    charSetID                       [2] CharSetID,
    data                            [3] OCTET STRING
                                        -- vCard format
}

BinaryEncoding                  ::= ENUMERATED
{
    base64Encoding                  (0),
    eightBit                        (1)
}

CodedEncoding                   ::= ENUMERATED
{
    sevenBit                        (0),
    base64Encoding                  (1),
    quotedPrintable                 (2),
    eightBit                        (3)
}

CompoundCompare                 ::= SEQUENCE
{
    operand1                        [0] SearchCondition,
    connection                      [1] Connection,
    operand2                        [2] SearchCondition
}

Connection                      ::= ENUMERATED
{
    andConnect                      (0),
    orConnect                       (1)
}
```

```
DataFormat                    ::= ENUMERATED
{
    vCard                         (0)
}

EntryData                     ::= SEQUENCE
{
    charSetID                     [0] CharSetID,
    data                          [1] OCTET STRING
                                      -- vCard format
}

EntryHandle                   ::= INTEGER

ExchangeDataFormat            ::= CHOICE
{
    vCard                         [0] VCardEncoding
}

GroupData                     ::= SET OF EntryData

GroupDescription              ::= SEQUENCE
{
    groupName                     [0] DisplayString,
    writable                      [1] BOOLEAN                    DEFAULT FALSE
                                      -- TRUE shows Group is writable
}

GroupHandle                   ::= INTEGER

GroupHandleList               ::= SET OF GroupHandle

GroupList                     ::= SET OF GroupDescription

HowToCompare                  ::= ENUMERATED
{
    equal                         (0),
    notEqual                      (1),
    greaterThan                   (2),
    greaterThanOrEqualTo          (3),
    lessThan                      (4),
    lessThanOrEqualTo             (5)
}

NumberOfActiveEntries         ::= INTEGER
```

```
PersonalityProtocol             ::= ENUMERATED
{
    SalutationAddressBook           (1),
    others                          (127)
}

SearchCondition                 ::= CHOICE
{
    simpleFieldCompare              [0] SimpleFieldCompare,
    compoundCompare                 [1] CompoundCompare
}

SearchHandle                    ::= INTEGER

SearchSupport                   ::= BOOLEAN

SimpleFieldCompare              ::= SEQUENCE
{
    howToCompare                    [0] HowToCompare,
    fieldName                       [1] DisplayString,
                                        -- Field Name (Parameter, Parameter,..)
                                        -- Encoded by 8859-1 (US ASCII) character set
    value                           [2] DisplayString
                                        -- Only String data can be searched.
                                        -- Encoded by the specified character set
}

Sort                            ::= ENUMERATED
{
    ascendingBitOrder               (0),
    descendingBitOrder              (1),
    weightingFactor                 (2),
    others                          (127)
}

SortSupport                     ::= BOOLEAN

VCardEncoding                   ::= SEQUENCE
{
    codedEncoding                   [0] CodedEncoding,
    binaryEncoding                  [1] BinaryEncoding
}
```

7.Message

7.1.Message Header

```
MsgHeader                    ::= SEQUENCE       -- The following components are
{                                               -- included in every message.
    msgSeqId                     INTEGER
                                     -- >0 : client initiating message sequence
                                     -- <0 : server initiating message sequence
    -- WARNING : Other components may be added here in future versions of architecture.
    -- Implementations should take this enhancement possibility into account.
}
```

7.2.Common

7.2.1.ACK, NACK

```
ACK                          ::= [APPLICATION tagACK] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    parameter1                   [0] ANY OPTIONAL,
    parameter2                   [1] ANY OPTIONAL,
    parameter3                   [2] ANY OPTIONAL
    --   :                                 :
    -- The number and data type of parameters depend on the associated command, and are defined
    -- by the specification of each associated command.
}
```

```
NACK     ::= [APPLICATION tagNACK] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    returnCode                   [0] ReturnCode,
    descriptor                   [1] OCTET STRING        OPTIONAL
                                     -- Additional information for the reason of rejection.
                                     -- Debug/diagnostics purpose. May be ignored.
}
```

7.2.2.Data Transfer

```
RequestDataTransfer          ::= [APPLICATION tagRequestDataTransfer] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    dataHandle                   [0] DataHandle OPTIONAL
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidDataHandle	dataHandle is unknown or missing when required	128
rcDataTransferAborted	Data transfer aborted by a sender	129

```
DataBlockDescription            ::= [APPLICATION tagDataBlockDescription] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    dataDescriptor                  [0] CHOICE
    {
        document                    [0] DocumentDataDescriptor,
        file                        [1] FileData
    }
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidDataDescriptor	dataDescriptor is incorrect or not supported	128
rcDataTransferAborted	Data transfer aborted by a receiver	129

```
TransferDataBlock               ::= [APPLICATION tagTransferDataBlock] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    beginDataBlock                  [0] BOOLEAN,
    endDataBlock                    [1] BOOLEAN,
    lastSegment                     [2] BOOLEAN,  -- TRUE in the last data block segment of the
                                                  -- last data block of “data”
    dataBlockBody                   [3] OCTET STRING
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidBeginEndFlag	begin/endDataBlock flag is out of sequence endDataBlock=FALSE when lastSegment=TRUE	128
rcInvalidDataBlockBody	Data content is incorrect or not supported (Note: applicable only when data content is immediately examined by the data receiver)	129
rcDataTransferAborted	Data transfer aborted by a receiver	130

```
RequestNextData             ::= [APPLICATION tagRequestNextData] SEQUENCE
{
                                COMPONENTS OF MsgHeader
}
```

NACK Response

Name	Description	ReturnCode
rcDataTransferAborted	Data transfer aborted by a sender	128

7.2.3.Attribute Repository

```
GetPrivateAttribute         ::= [APPLICATION tagGetPrivateAttribute] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    attributeIdList             [0] SET OF AttributeID
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	AttributeList	

NACK Response

No message-specific return code

```
GetGlobalAttribute          ::= [APPLICATION tagGetGlobalAttribute] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    attributeIdList             [0] SET OF AttributeID
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	AttributeList	

NACK Response

No message-specific return code

```
SetPrivateAttribute          ::= [APPLICATION tagSetPrivateAttribute] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    attributeList                [0] SET OF SEQUENCE
    {
        attributeId              [0] AttributeID,
        attributeValue           [1] ANY            OPTIONAL
                                     -- Type is defined by each attribute.
                                     -- If omitted, attribute is deleted.
    }
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidAttributeId	attributeId is incorrect or not supported	128
rcInvalidAttributeValue	attributeValue is incorrect or not supported	129

7.2.4.Job-Related

```
QueryJobStatus               ::= [APPLICATION tagQueryJobStatus] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    jobHandle                    [0] JobHandle
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	JobStatusCode	
parameter2	ReasonCode	Optional

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128

```
QueryJobEntryStatus          ::= [APPLICATION tagQueryJobEntryStatus] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    jobHandle                    [0] JobHandle,
    jobEntryId                   [1] JobEntryID    OPTIONAL
                                     -- If omitted, the status of all the job entries is requested.
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	JobStatusCode, or JobEntriesStatus	
parameter2	ReasonCode	Optional

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcInvalidJobEntryId	jobEntryId is unknown	129

```
NotifyJobStatus              ::= [APPLICATION tagNotifyJobStatus] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    jobHandle                    [0] JobHandle,
    jobStatusCode                [1] JobStatusCode,
    reasonCode                   [2] ReasonCode              OPTIONAL
                                     -- present only if jobStatusCode=suspended or error
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is incorrect or unknown	128
rcInvalidJobStatusCode	jobStatusCode is incorrect or not supported	131
rcInvalidReasonCode	reasonCode is incorrect or not supported or missing when required	132

```
NotifyJobEntryStatus           ::= [APPLICATION tagNotifyJobEntryStatus] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
    jobHandle                      [0] JobHandle,
    jobEntryId                     [1] JobEntryID,
    jobStatusCode                  [2] JobStatusCode,
    reasonCode                     [3] ReasonCode           OPTIONAL
                                       -- present only if jobStatusCode=suspended or error
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is incorrect or unknown	128
rcInvalidJobEntryId	jobEntryId is incorrect or unknown	129
rcInvalidJobStatusCode	jobStatusCode is incorrect or not supported	131
rcInvalidReasonCode	reasonCode is incorrect or not supported or missing when required	132

```
ChangeJobAttribute             ::= [APPLICATION tagChangeJobAttribute] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
    jobHandle                      [0] JobHandle,
    attributeId                    [1] AttributeID,
    attributeValue                 [2] ANY
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcJobAlreadyExecuted	Job is being executed (or completed) and it is too late to change the attribute value	130
rcInvalidAttributeId	attributeId is incorrect or not supported	133
rcInvalidAttributeValue	attributeValue is incorrect or not supported	134

```
ChangeJobEntryAttribute          ::= [APPLICATION tagChangeJobEntryAttribute] SEQUENCE
{
                                      COMPONENTS OF MsgHeader,
    jobHandle                         [0] JobHandle,
    jobEntryId                        [1] JobEntryID,
    attributeId                       [2] AttributeID,
    attributeValue                    [3] ANY
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcInvalidJobEntryId	jobEntryId is unknown	129
rcJobAlreadyExecuted	Job is being executed (or completed) and it is too late to change the attribute value	130
rcInvalidAttributeId	attributeId is incorrect or not supported	133
rcInvalidAttributeValue	attributeValue is incorrect or not supported	134

```
SuspendJob                       ::= [APPLICATION tagSuspendJob] SEQUENCE
{
                                      COMPONENTS OF MsgHeader,
    jobHandle                         [0] JobHandle
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcJobAlreadyExecuted	Job has already been executed	130

```
SuspendJobEntry                  ::= [APPLICATION tagSuspendJobEntry] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
   jobHandle                         [0] JobHandle,
   jobEntryId                        [1] JobEntryID
}
```

ACK Response

No parameter

NACK Response

Name	**Description**	**ReturnCode**
rcInvalidJobHandle	jobHandle is unknown	128
rcInvalidJobEntryId	jobEntryId is unknown	129
rcJobAlreadyExecuted	Job entry has already been executed	130

```
ResumeJob                        ::= [APPLICATION tagResumeJob] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
   jobHandle                         [0] JobHandle
}
```

ACK Response

No parameter

NACK Response

Name	**Description**	**ReturnCode**
rcInvalidJobHandle	jobHandle is unknown	128
rcJobAlreadyExecuted	Job has already been executed	130

```
ResumeJobEntry                   ::= [APPLICATION tagResumeJobEntry] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
   jobHandle                         [0] JobHandle,
   jobEntryId                        [1] JobEntryID
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcInvalidJobEntryId	jobEntryId is unknown	129
rcJobAlreadyExecuted	Job entry has already been executed	130

```
CancelJob                       ::= [APPLICATION tagCancelJob] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
    jobHandle                        [0] JobHandle,
    abort                            [1] BOOLEAN
                                         -- if TRUE, job is canceled either queued or being executed
                                         -- if FALSE, job is canceled only if execution has not started
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcJobAlreadyExecuted	Job has already been executed	130

```
CancelJobEntry                  ::= [APPLICATION tagCancelJobEntry] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
    jobHandle                        [0] JobHandle,
    jobEntryId                       [1] JobEntryID,
    abort                            [2] BOOLEAN
                                         -- if TRUE, job is canceled either queued or being executed
                                         -- if FALSE, job is canceled only if execution has not started
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcInvalidJobEntryId	jobEntryId is unknown	129
rcJobAlreadyExecuted	Job entry has already been executed	130

```
FreeJobHandle                ::= [APPLICATION tagFreeJobHandle] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    jobHandle                    [0] JobHandle
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcJobNotCompleted	Job execution has not completed yet	135

```
StartMonitorJobStatus        ::= [APPLICATION tagStartMonitorJobStatus] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    jobHandle                    [0] JobHandle,
    jobStatusNotificationMode    [1] JobStatusNotificationMode
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128
rcInvalidJobStatusNotificationMode	jobStatusNotificationMode is incorrect	136

```
CancelMonitorJobStatus          ::= [APPLICATION tagCancelMonitorJobStatus] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    jobHandle                       [0] JobHandle
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidJobHandle	jobHandle is unknown	128

7.2.5.Dynamic Status

```
QueryDynamicStatus              ::= [APPLICATION tagQueryDynamicStatus] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    dynamicStatusId                 [0] DynamicStatusID
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	ANY	

NACK Response

Name	Description	ReturnCode
rcInvalidDynamicStatusId	dynamicStatusId is incorrect or not supported	129

```
SubscribeEvent                  ::= [APPLICATION tagSubscribeEvent] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    eventList                       [0] SET OF DynamicStatusID,
    life                            [1] Life,
    checkInterval                   [2] INTEGER             OPTIONAL
                                        -- Interval (in seconds) for the FU-side SLM to periodically
                                        -- check the availability of the [Client] FU to receive the job
                                        -- status notification
                                        -- This parameter shall be present if life = persistent.
                                        -- This parameter shall be omitted if life = session.
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	SubscriptionHandle	

NACK Response

Name	Description	ReturnCode
rcInvalidDynamicStatusId	eventList contains one or more incorrect or unsupported dynamicStatusId	129
rcInvalidLife	life is incorrect or not supported	131
rcInvalidCheckInterval	checkInterval is incorrect or not supported	132

```
NotifyEvent                    ::= [APPLICATION tagNotifyEvent] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
    subscriptionHandle             [0] SubscriptionHandle,
    dynamicStatusId                [1] DynamicStatusID,
    dynamicStatusValue             [2] ANY
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidSubscriptionHandle	subscriptionHandle is unknown	128
rcInvalidDynamicStatusId	dynamicStatusId is incorrect or not supported	129
rcInvalidDynamicStatusValue	dynamicStatusValue is incorrect or not supported	130

```
UnsubscribeEvent               ::= [APPLICATION tagUnsubscribeEvent] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
    subscriptionHandle             [0] SubscriptionHandle
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidSubscriptionHandle	subscriptionHandle is unknown	128

7.2.6.Vendor Escape

```
VendorEscape                ::= [APPLICATION tagVendorEscape] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    parameter                   [0] ANY
}
```

7.3.Document Systems

7.3.1.[Print] Functional Unit

```
Print                       ::= [APPLICATION tagPrint] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
    modeOfDataTransfer          [0] DataTransferMode             OPTIONAL,
                                    -- Override Global/Private Attribute
    dataSource                  [1] DataLocation                 DEFAULT client,
    dataHandle                  [2] DataHandle                   OPTIONAL,
                                    -- Omitted in immediate mode data transfer from client, or
                                    -- if the source data location is specified by URL
    inputDocumentFormat         [3] DocumentDataDescriptor       OPTIONAL,
                                    -- Present if and only if dataSource = url
    life                        [4] Life                         DEFAULT job,
                                    -- Specify how long FU should keep a job status:
                                    -- for job life or for session life or persistently.
    jobStatusNotificationMode   [5] JobStatusNotificationMode    OPTIONAL,
                                    -- If omitted, no notification is made.
    notificationScheme          [6] NotificationScheme           OPTIONAL,
                                    -- Omitted unless the job status notifications are to be
                                    -- sent to a [Client] FU other than the client that is
                                    -- sending this command
    printControlAttribute       [7] PrintControlAttribute        OPTIONAL
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	JobHandle	

NACK Response

Name	Description	ReturnCode
rcInvalidModeOfDataTransfer	modeOfDataTransfer is incorrect or not supported	129
rcInvalidDataSource	dataSource is incorrect or not supported	130
rcInvalidDataHandle	dataHandle is incorrect	131
rcInvalidInputDocumentFormat	inputDocumentFormat is incorrect or not supported	132
rcInvalidLife	life is incorrect or not supported	133
rcInvalidJobStatusNotificationMode	jobStatusNotificationMode is incorrect or not supported	134
rcInvalidNotificationScheme	notificationScheme is incorrect or not supported	135
rcInvalidPaperSize	printPaperSize is incorrect or not supported	136
rcInvalidResolution	printResolution is incorrect or not supported	137
rcInvalidPaperDirection	printPaperDirection is incorrect or not supported	138
rcInvalidCopyCount	printCopyCount is incorrect or not supported	139
rcInvalidPaperInputSelect	printPaperInputSelect is incorrect or not supported	140
rcInvalidPaperOutputSelect	printPaperOutputSelect is incorrect or not supported	141
rcInvalidOutputBinSelect	printOutputBinSelect is incorrect or not supported	142
rcInvalidDuplexMode	printDuplexMode is incorrect or not supported	143
rcInvalidFaceUpMode	printFaceUpMode is incorrect or not supported	144
rcInvalidPriority	printPriority is incorrect or not supported	145
rcInvalidStaplingSelect	printStaplingSelect is incorrect or not supported	146

Job-Specific ReasonCode

Name	Description	ReasonCode
suspendedByClientRequest	suspended by SuspendJob command	128
temporaryBusy	suspended due to equipment temporary busy.	129
waitingForRetry	in waiting mode for retry.	130
retryOut	terminated due to retry out of printing attempts.	131
printerError	terminated due to equipment detected errors, e.g., noPaper, noToner, and etc..	132

```
ListPrintJob                    ::= [APPLICATION tagListPrintJob] SEQUENCE
{
                                    COMPONENTS OF MsgHeader
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcNoJob	there is no job	128

7.3.2.[FAX Data Send] Functional Unit

```
SendFAX                         ::= [APPLICATION tagSendFAX] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    modeOfDataTransfer              [0] DataTransferMode          OPTIONAL,
                                        -- Override Global / Private Attribute
    dataSource                      [1] DataLocation              DEFAULT client,
    dataHandle                      [2] DataHandle                OPTIONAL,
                                        -- Omitted in immediate mode data transfer from client, or
                                        -- if the source data location is specified by URL
    inputDocumentFormat             [3] DocumentDataDescriptor    OPTIONAL,
                                        -- Present if and only if dataSource = url
    life                            [4] Life                      DEFAULT job,
                                        -- Specify how long FU should keep a job status:
                                        -- for job life or for session life or persistently.
    jobStatusNotificationMode       [5] JobStatusNotificationMode OPTIONAL,
                                        -- If omitted, no notification is made.
    notificationScheme              [6] NotificationScheme        OPTIONAL,
                                        -- Omitted unless the job status notifications are to be
                                        -- sent to a [Client] FU other than the client that is
                                        -- sending this command
    faxControlAttribute             [7] FaxControlAttribute
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	JobHandle	

NACK Response

Name	Description	ReturnCode
rcInvalidModeOfDataTransfer	modeOfDataTransfer is incorrect or not supported	128
rcInvalidDataSource	dataSource is incorrect or not supported	129
rcInvalidDataHandle	dataHandle is incorrect	130
rcInvalidInputDocumentFormat	inputDocumentFormat is incorrect or not supported	131
rcInvalidLife	life is incorrect or not supported	132
rcInvalidJobStatusNotificationMode	jobStatusNotificationMode is incorrect or not supported	133
rcInvalidNotificationScheme	notificationScheme is incorrect or not supported	134
rcInvalidCoverSheetGen	coverSheetGen is incorrect or not supported	135
rcInvalidPageHeaderGen	pageHeaderGen is incorrect or not supported	136
rcTooManyCalledSubscribers	The number of called subscribers exceeds the limit	137
rcInvalidFaxNumber	faxNumber is incorrect	138
rcInvalidSubAddressNumber	subAddressNumber is incorrect	139
rcInvalidFaxProtocol	faxProtocol is incorrect or not supported	140
rcInvalidOrderingData	orderingData is incorrect or not supported	141
rcInvalidRequestPriority	requestPriority is incorrect or not supported	142
rcInvalidRetryCount	retryCount is incorrect or not supported	143

Job-Specific ReasonCode

Name	Description	ReasonCode
timeOut	time-out detected during get-line. (When zero is specified in retryCount)	128
retryOut	terminated due to retry out. (When zero is specified for retryCount, this parameter is not returned. Instead calledSubscriberBusy or timeOut is returned,)	129
calledSubscriberBusy	busy status detected for called subscriber.	130
modemShiftDownFailed	connection failed with the lowest speed.	131
callSetUpFailed	call setup failed.	132
negotiationFailed	negotiation failed.	133
notReceiveExpectedFrame	expecting frame(s) not received on G3 protocol.	134
receiveUnexpectedFrame	unexpected frame(s) received on G3 protocol.	135
thirdTryFail	retried-out during G3 protocol.	136
waitingForRetry	in waiting mode for retry.	137

```
ListFaxJob                    ::= [APPLICATION tagListFaxJob] SEQUENCE
{
                                 COMPONENTS OF MsgHeader
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcNoJob	there is no job	128

7.3.3.[DOC Storage] Functional Unit

```
RetrieveDoc                 ::= [APPLICATION tagRetrieveDoc] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   folderId                     [0] FolderID,
   documentId                   [1] DocumentID,
   dataDestination              [2] DataLocation          DEFAULT client,
   startDataBlock               [3] INTEGER               DEFAULT 1,
                                   -- If omitted, the document is retrieved
                                   -- from the first data block.
   endDataBlock                 [4] INTEGER               OPTIONAL
                                   -- If omitted, the document is retrieved
                                   -- through the last data block.
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	DataHandle	Optional

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcAccessRejected	access is not authorized for the user	130
rcInvalidDataDestination	dataDestination is incorrect or not supported	131
rcInvalidStartDataBlock	startDataBlock is incorrect	132
rcInvalidEndDataBlock	endDataBlock is incorrect	133

```
StoreDoc                    ::= [APPLICATION tagStoreDoc] SEQUENCE
{
                              COMPONENTS OF MsgHeader,
   folderId                   [0] FolderID,
   dataSource                 [1] DataLocation             DEFAULT client,
   dataHandle                 [2] DataHandle               OPTIONAL,
                                 -- Exists only if dataSource=functionalUnit
   modeOfStore                [3] DataStoreMode            OPTIONAL,
                                 -- Override Global / Private Attribute
   inputDocumentFormat        [4] DocumentDataDescriptor   OPTIONAL,
                                 -- Present if and only if dataSource = url
   ownerName                  [5] OwnerName                OPTIONAL,
   docComment                 [6] DocComment               OPTIONAL,
   typeOfContent              [7] DataContent              OPTIONAL
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	DocumentID	

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown	128
rcAccessRejected	access is not authorized for the user	130
rcInvalidDataSource	dataSource is incorrect or not supported	131
rcInvalidDataHandle	dataHandle is unknown	132
rcInvalidModeOfStore	modeOfStore is incorrect or not supported	133
rcInvalidInputDocumentFormat	inputDocumentFormat is incorrect or not supported	134
rcStorageFull	storage is full	135
rcInvalidTypeOfContent	typeOfContent is incorrect or not supported	136

```
DeleteDoc                   ::= [APPLICATION tagDeleteDoc] SEQUENCE
{
                              COMPONENTS OF MsgHeader,
   folderId                   [0] FolderID,
   documentId                 [1] DocumentID
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcAccessRejected	access is not authorized for the user	130

```
CopyDoc                          ::= [APPLICATION tagCopyDoc] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
    sourceFolder                     [0] FolderID,
    documentId                       [1] DocumentID,
    destinationFolder                [2] FolderID                    OPTIONAL,
                                     -- if omitted, the same as sourceFolder
    updateDateTime                   [3] BOOLEAN                     DEFAULT FALSE
                                     -- if TRUE, update the document's
                                     -- creationDateTime with the current time.
                                     -- if FALSE or omitted, use the document's
                                     -- old creationDateTime.
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	DocumentID	

NACK Response

Name	Description	ReturnCode
rcInvalidSourceFolderId	sourceFolder is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcSourceAccessRejected	access to the source folder/document is not authorized for the user	130
rcInvalidDestinationFolderId	destinationFolder is unknown	131
rcDestinationAccessRejected	access to the destination folder is not authorized for the user	132

```
MoveDoc                    ::= [APPLICATION tagMoveDoc] SEQUENCE
{
                               COMPONENTS OF MsgHeader,
   sourceFolder                [0] FolderID,
   documentId                  [1] DocumentID,
   destinationFolder           [2] FolderID               OPTIONAL,
                               -- if omitted, the same as sourceFolder
   updateDateTime              [3] BOOLEAN                DEFAULT FALSE
                               -- if TRUE, update the document's
                               -- creationDateTime with the current time.
                               -- if FALSE or omitted, use the document's
                               -- old creationDateTime.
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	DocumentID	

NACK Response

Name	Description	ReturnCode
rcInvalidSourceFolderId	sourceFolder is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcSourceAccessRejected	access to the source folder/document is not authorized for the user	130
rcInvalidDestinationFolderId	destinationFolder is unknown	131
rcDestinationAccessRejected	access to the destination folder is not authorized for the user	132

```
ChangeDocDesc              ::= [APPLICATION tagChangeDocDesc] SEQUENCE
{
                               COMPONENTS OF MsgHeader,
   folderId                    [0] FolderID,
   documentId                  [1] DocumentID,
   ownerName                   [2] OwnerName       OPTIONAL,
   docComment                  [3] DocComment      OPTIONAL
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown	128
rcInvalidDocumentId	documentId is unknown	129
rcAccessRejected	access is not authorized for the user	130

```
CreateFolder                ::= [APPLICATION tagCreateFolder] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   ownerName                    [0] OwnerName                 OPTIONAL,
   folderComment                [1] FolderComment             OPTIONAL
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	FolderID	

NACK Response

No message-specific return code

```
ChangeFolderDesc            ::= [APPLICATION tagChangeFolderDesc] SEQUENCE
{
                                COMPONENTS OF MsgHeader,
   folderId                     [0] FolderID,
   ownerName                    [1] OwnerName                 OPTIONAL,
   folderComment                [2] FolderComment             OPTIONAL
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown or incorrect (0)	128
rcAccessRejected	access is not authorized for the user	130

```
ListFolder                    ::= [APPLICATION tagListFolder] SEQUENCE
{
                                  COMPONENTS OF MsgHeader
}
```

ACK Response

No parameter

NACK Response

No message-specific return code

```
DeleteFolder                  ::= [APPLICATION tagDeleteFolder] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   folderId                       [0] FolderID
                                  -- Folder should be empty before deleted.
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown or incorrect (0)	128
rcAccessRejected	access is not authorized for the user	130
rcFolderNotEmpty	folder contains document(s)	131

```
ListFolderDoc                 ::= [APPLICATION tagListFolderDoc] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   folderId                       [0] FolderID
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidFolderId	folderId is unknown or incorrect (0)	128
rcAccessRejected	access is not authorized for the user	130

7.4.Voice Message Systems

7.4.1.[Voice Message Storage] Functional Unit

```
ListFolderContentVM          ::= [APPLICATION tagListFolderContentVM] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
   folderId                      [0] FolderID
}
```

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	138
rcFolderAccessRejected	Access to folder is not authorized	139

```
SendVM                       ::= [APPLICATION tagSendVM] SEQUENCE
{                                COMPONENTS OF MsgHeader,
   folderId                      [0] FolderID,
   voiceMsgId                    [1] VoiceMsgID,
   recipients                    [2] SET OF Recipient,
   deliveryGrade                 [3] DeliveryGrade        OPTIONAL,
                                    -- sender specifies the grade of delivery. This information
                                    -- is for the mail server
   priorityLevel                 [4] SimpleJobPriority    OPTIONAL,
                                    -- sender specifies the priority level of the message. This
                                    -- information is for the receiver
   subject                          [5] DisplayString               OPTIONAL
                                    -- sender specifies the subject of the message.
                                    -- maximum 256 characters
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	JobHandle	

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	138
rcFolderAccessRejected	Access to folder is not authorized	139
rcInvalidVoiceMsgId	Specified Voice Message not found	148
rcInvalidRecipientId	Specified recipientId is invalid	168
rcInvalidRecipientType	Specified recipientType is invalid	169
rcInvalidDeferredDeliveryTime	Specified deferredDeliveryTime is invalid	188
rcInvalidDeliveryGrade	Specified delivery grade not valid	189
rcInvalidPriorityLevel	Specified priority level not valid	190
rcInvalidSubject	Specified subject not valid	191

```
PlayVM                        ::= [APPLICATION tagPlayVM] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   folderId                       [0] FolderID,
   voiceMsgId                     [1] VoiceMsgID,
   receivers                      [2] SET OF Receiver,
   headerInformation              [3] HeaderInformation     OPTIONAL,
   voiceDuration                  [4] INTEGER               OPTIONAL,
   voiceSpeed                     [5] INTEGER               OPTIONAL,
   voiceVolume                    [6] INTEGER               OPTIONAL
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	JobHandle	

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	138
rcFolderAccessRejected	Access to folder is not authorized	139
rcInvalidVoiceMessageId	Specified Voice Message not found	148
rcInvalidReceiver	Specified receiver is not valid	170
rcInvalidDeferredDeliveryTime	Specified deferredDeliveryTime is not valid	188
rcInvalidHeaderInfo	Specified header information not valid	192
rcInvalidVoiceDuration	Specified voice duration not valid	200

rcInvalidVoiceSpeed	Specified voice speed not valid	201
rcInvalidVoiceVolume	Specified voice volume not valid	202

Job-Specific ReasonCode

Name	Description	ReasonCode
equipmentError	terminated due to equipment detected errors.	128
waitingForRetry	in waiting mode for retry call.	129

```
SynthesizeVM                    ::= [APPLICATION tagSynthesizeVM] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
   folderId                         [0] FolderID,
   text                             [1] DisplayString,
   textLanguage                     [2] TextLanguage          OPTIONAL,
   voiceMessageDataDescriptor       [3] VoiceMessageDataDescriptor    OPTIONAL,
   voiceType                        [4] VoiceType             OPTIONAL,
   voiceSpeed                       [5] INTEGER               OPTIONAL,
   voiceVolume                      [6] INTEGER               OPTIONAL
}
```

ACK Response

Parameter Name	Data Type	Note
parameter1	VoiceMsgID	

NACK Response

Name	Description	ReturnCode
rcFolderNotFound	Specified folder does not exist	138
rcFolderAccessRejected	Access to folder is not authorized	139
rcInvalidText	Text data is not valid	218
rcInvalidTextLanguage	Text language is not valid	210
rcInvalidEncodingAlgo	Specified encoding algorithm not valid	203
rcInvalidSamplingRate	Specified sampling rate not valid	204
rcInvalidVoiceType	Specified voice type not valid	205
rcInvalidVoiceMessageDescriptor	Specified VoiceMessageDescriptor not valid	206
rcInvalidVoiceMessageDataFormat	Specified VoiceMessageDataFormat not valid	207
rcInvalidVoiceMessageFormatInterpretation	Specified VoiceMessageFormatInterpretation not valid	208
rcInvalidVoiceSpeed	Specified voice speed not valid	201
rcInvalidVoiceVolume	Specified voice volume not valid	202

```
ListVMSJob                    ::= [APPLICATION tagListVMSJob] SEQUENCE
{
                                  COMPONENTS OF MsgHeader
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcNoJob	There is no job	128

7.5.Personal Information Systems

7.5.1. [Address Book] Functional Unit

```
ListGroups                    ::= [APPLICATION tagListGroups] SEQUENCE
{
                                  COMPONENTS OF MsgHeader
}
```

NACK Response

No message-specific return code

```
OpenGroup                     ::= [APPLICATION tagOpenGroup] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
   groupName                      [0] DisplayString,
   readWriteAccess                [1] BOOLEAN              DEFAULT FALSE
                                      -- TRUE shows Writable access
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	GroupHandle	
parameter-2	ReturnCode (rcBeingModified, ENUMERATED (148))	The Group is being opened for read/write operation by another client.

NACK Response

Name	Description	ReturnCode
rcNoGroup	There is not specified Group.	138
rcInvalidAccessMode	Access mode is not valid.	139
rcCanNotBeOpened	The Group can not be opened.	140
rcBeingModified	The Group is already opened for read/write operation	148

```
CloseGroup                    ::= [APPLICATION tagCloseGroup] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
    groupHandle                   [0] INTEGER
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcInvalidGroupHandle	Specified Group Handle is invalid	141

```
CreateGroup                   ::= [APPLICATION tagCreateGroup] SEQUENCE
{
                                  COMPONENTS OF MsgHeader,
    groupName                     [0]DisplayString
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	GroupHandle	

NACK Response

Name	Description	ReturnCode
rcGroupAlreadyExist	Specified Group name already exists in an FU.	144

```
DeleteGroup                  ::= [APPLICATION tagDeleteGroup] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    groupHandle                  [0] INTEGER
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcGroupHasEntry	Specified Group has an Entry.	145
rcInvalidGroupHandle	Specified Group Handle is invalid.	141

```
RenameGroup                  ::= [APPLICATION tagRenameGroup] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    groupHandle                  [0] INTEGER,
    newName                      [1] DisplayString
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcGroupAlreadyExist	Specified Group name already exists in an FU.	144
rcOperationNotPermitted	Operation to the Group is not permitted.	146

```
GetGroupData                 ::= [APPLICATION tagGetGroupData] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    groupHandle                  [0] INTEGER,
    exchangeDataFormat           [1] ExchangeDataFormat
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidExchangeDataFormat	Specified exchange data format is invalid.	168
rcExchangeDataFormatNotSupported	Specified exchange data format is not supported.	169
rcCodedEncodingNotSupported	Specified Coded encoding is not supported.	170
rcBinaryEncodingNotSupported	Specified Binary encoding is not supported.	171

```
ListActiveEntries                ::= [APPLICATION tagListActiveEntries] SEQUENCE
{
                                     COMPONENTS OF MsgHeader
}
```

NACK Response

Name	Description	ReturnCode
rcNoActiveEntries	There is no active Entries.	178
rcCommandNotSupported	Command for Entry Operation is not supported.	218

```
GetEntryData                     ::= [APPLICATION tagGetEntryData] SEQUENCE
{
                                     COMPONENTS OF MsgHeader,
    groupHandle                      [0] INTEGER,
    entryHandle                      [1] INTEGER,
    exchangeDataFormat               [2] ExchangeDataFormat
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidGroupHandle	Specified Group Handle is invalid	141
rcInvalidEntryHandle	Specified Entry Handle is invalid	179
rcInvalidExchangeDataFormat	Specified exchange data format is invalid	168
rcExchangeDataFormatNotSupported	Specified exchange data format is not supported.	169
rcCodedEncodingNotSupported	Specified Coded encoding is not supported.	170
rcBinaryEncodingNotSupported	Specified Binary encoding is not supported.	171
rcCommandNotSupported	Command for Entry Operation is not supported.	218

```
GetActiveEntryData              ::= [APPLICATION tagGetActiveEntryData] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    position                        [0] INTEGER,
    exchangeDataFormat              [1] ExchangeDataFormat
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidPosition	Specified position is invalid.	180
rcNoActiveEntries	There is no active Entries.	178
rcInvalidExchangeDataFormat	Specified exchange data format is invalid.	168
rcExchangeDataFormatNotSupported	Specified exchange data format is not supported.	169
rcCodedEncodingNotSupported	Specified Coded encoding is not supported.	170
rcBinaryEncodingNotSupported	Specified Binary encoding is not supported.	171
rcCommandNotSupported	Command for Entry Operation is not supported.	218

```
AddEntryData                    ::= [APPLICATION tagAddEntryData] SEQUENCE
{
                                    COMPONENTS OF MsgHeader,
    groupHandle                     [0] INTEGER,
    charSetID                       [1] CharSetID,
    dataFormat                      [2] DataFormat,
    data                            [3] OCTET STRING
                                        -- vCard format
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	EntryHandle	

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcNoRoomToAddReplace	The Group has no room to add an Entry data.	147
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidDataFormat	Specified data format is invalid or not supported.	167
rcInvalidReceivedDataFormat	Received data format is invalid.	181
rcCharacterSetNotSupported	Specified Character set is not supported	172
rcCommandNotSupported	Command for Entry Operation is not supported.	218

```
DeleteEntryData              ::= [APPLICATION tagDeleteEntryData] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    groupHandle                  [0] INTEGER,
    entryHandle                  [1] INTEGER
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidEntryHandle	Specified Entry Handle is invalid.	179
rcCommandNotSupported	Command for Entry Operation is not supported.	218

```
ReplaceEntryData             ::= [APPLICATION tagReplaceEntryData] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
    groupHandle                  [0] INTEGER,
    entryHandle                  [1] INTEGER,
    charSetID                    [2] CharSetID,
    dataFormat                   [3] DataFormat,
    data                         [4] OCTET STRING
                                     -- vCard format
}
```

ACK Response

No parameter

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcNoRoomToAddReplace	The Group has no room to replace an Entry data.	147
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidDataFormat	Specified data format is invalid or not supported.	167
rcInvalidReceivedDataFormat	Received data format is invalid.	181
rcCharacterSetNotSupported	Specified Character set is not supported	172
rcCommandNotSupported	Command for Entry Operation is not supported.	218
rcInvalidEntryHandle	Specified Entry Handle is invalid.	179

```
MoveEntryData                  ::= [APPLICATION tagMoveEntryData] SEQUENCE
{
                                   COMPONENTS OF MsgHeader,
    fromGroupHandle                [0] INTEGER,
    fromEntryHandle                [1] INTEGER,
    toGroupHandle                  [2] INTEGER
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	EntryHandle	

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcInvalidFromGroupHandle	Specified ‘from’ Group Handle is invalid.	142
rcInvalidToGroupHandle	Specified ‘to’ Group Handle is invalid.	143
rcInvalidEntryHandle	Specified Entry Handle is invalid.	179
rcCommandNotSupported	Command for Entry Operation is not supported.	218

```
CopyEntryData                ::= [APPLICATION tagCopyEntryData] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
   fromGroupHandle               [0] INTEGER,
   fromEntryHandle               [1] INTEGER,
   toGroupHandle                 [2] INTEGER
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	EntryHandle	

NACK Response

Name	Description	ReturnCode
rcOperationNotPermitted	Operation to the Group is not permitted.	146
rcInvalidFromGroupHandle	Specified ‘from’ Group Handle is invalid.	142
rcInvalidToGroupHandle	Specified ‘to’ Group Handle is invalid.	143
rcInvalidEntryHandle	Specified Entry Handle is invalid.	179
rcInvalidGroupHandle	Specified to Group Handle is invalid.	141
rcCommandNotSupported	Command for Entry Operation is not supported.	218

```
SearchFieldData              ::= [APPLICATION tagSearchFieldData] SEQUENCE
{
                                 COMPONENTS OF MsgHeader,
   searchHandle                  [0] INTEGER,
   charSetID                     [1] CharSetID,
                                     -- to specify the Entry which data is encoded by this character set
   codedEncoding                 [2] CodedEncoding,
                                     -- Value to be compare is encoded by specified encoding.
   searchCondition               [3] SearchCondition,
   groupHandleList               [4] GroupHandleList               OPTIONAL
                                     -- Optional for search operation to current Active Entries
}
```

ACK Response

Parameter Name	Data Type	Note
parameter-1	SearchHandle	
parameter-2	NumberOfActiveEntries	

NACK Response

Name	Description	ReturnCode
rcInvalidGroupHandle	Specified Group Handle is invalid.	141
rcInvalidSearchHandle	Specified Search Handle is invalid.	182
rcInvalidCharacterSet	Specified Character set is not same as current active Entries.	173
rcCodedEncodingNotSupported	Specified Coded encoding is not supported.	170
rcDataNotFound	Data not found.	183
rcCharacterSetNotSupported	Specified Character set is not supported	172
rcCommandNotSupported	Command for Entry/Field Operation is not supported.	218

```
GetActiveEntriesFieldData          ::= [APPLICATION tagGetActiveEntriesFieldData] SEQUENCE
{
                                        COMPONENTS OF MsgHeader,
    fieldName                           [0] DisplayString,
                                            -- Field Name (Parameter, Parameter,..)
                                            -- Encoded by 8859-1 (US ASCII) character set
    codedEncoding                       [1] CodedEncoding,
    sort                                [2] Sort                    OPTIONAL
}
```

NACK Response

Name	Description	ReturnCode
rcInvalidFieldName	Specified Field name is invalid.	208
rcFieldDataNotFound	Specified Field data not found.	209
rcCodedEncodingNotSupported	Specified Coded encoding is not supported.	170
rcSortNoSupport	Sort operation is not supported.	210
rcNoActiveEntries	There is no active Entries.	178
rcCommandNotSupported	Command for Entry/Field Operation is not supported.	218

8.Functional Unit ID

Functional Unit Name	**ID**
Wild (for use with QueryCapability)	0
[Client]	1000
[Print]	10000
[Document Storage]	11000
[FAX Data Send]	12000
[FAX Data]	13000
[Voice Message Storage]	20000
[Address Book]	30000

9.Attribute & Dynamic Status ID

9.1.Range of Number Assignments

Functional Unit	Minimum	Maximum
Common to All Functional Unit Types	0	999
[Client]	1000	1999
[Print]	10000	10999
[DOC Storage]	11000	11999
[FAX Data Send]	12000	12999
[FAX Data]	13000	13999
[Voice Message Storage]	20000	20999
[Address Book]	30000	30999

9.2.Common

Capability Attribute Name	ID	Data Type	Compare Function ID
Major version	10	INTEGER	intEqualTo
Minor version	11	INTEGER	intGreaterThanOrEqualTo
Default coded character set	20	CharSetID	intEqualTo
FU name	30	DisplayString (SIZE(0..63))	strEqualTo
Manufacturer name	40	DisplayString (SIZE(0..63))	strEqualTo
Manufacturer product name	41	DisplayString (SIZE(0..63))	strEqualTo
Manufacturer product version	42	DisplayString (SIZE(0..63))	strEqualTo
Physical location	50	DisplayString (SIZE(0..255))	strEqualTo
Contact person name	51	DisplayString (SIZE(0..255))	strEqualTo
Authentication flavors	60	SET OF AuthenticationFlavor	setIntDoesContain

9.3.[Client] Functional Unit

Capability Attribute Name	ID	Data Type	Compare Function ID
User ID	1000	UserID	strEqualTo

9.4.Document Systems

9.4.1.[Print] Functional Unit

9.4.1.1.Capability and Command Attribute

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute (default[6])	Private/ Job Attribute
personalityProtocol	10000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No/No
supportedCommand	10001	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No/No
dynamicStatusId	10002	N/A	SET OF DynamicStatusID (setIntDoesContain)	No	No/No
spoolStorage	10003	N/A	SpoolStorage (boolEqualTo)	No	No/No
minimumCheckInterval -- the minimum allowed -- value to be set in the -- checkInterval parameter -- of a SubscribeEvent -- command	10004	N/A	INTEGER (intGreaterThanOrEqualTo)	No	No/No
documentFormat	10010	DataFormat	SET OF DataFormat (setIntDoesContain)	No	No/No
imageCompAlgorithm	10011	ImageCompAlgorithm	SET OF ImageCompAlgorithm (setIntDoesContain)	No	No/No
imageByteFillOrder	10012	ByteFillOrder	SET OF ByteFillOrder (setIntDoesContain)	No	No/No
imageResolution	10013	ImageResolution	SET OF ImageResolution (setIntDoesContain)	No	No/No
printPaperSize	10020	PaperSize	SET OF PaperSize (setIntDoesContain)	Yes	No/No
printResolution	10021	ImageResolution	SET OF ImageResolution (setIntDoesContain)	Yes	No/No
printPaperDirection	10022	PaperDirection	SET OF PaperDirection (setIntDoesContain)	Yes	No/No
printCopyCount	10023	INTEGER	INTEGER -- max value (intGreaterThanOrEqualTo)	No (1)	No/No
printPaperInputSelect	10024	PrintPaperInputSelect	SET OF PrintPaperInputSelect (setIntDoesContain)	Yes	No/No
printPaperOutputSelect	10025	PrintPaperOutputSelect	SET OF PrintPaperOutputSelect (setIntDoesContain)	Yes	No/No

[6] Implementation default values to be referred to when neither command parameter nor Private Attribute value is set.

printOutputBinSelect	10026	PrintOutputBinSelect	PrintOutputBinSelect —maximum bin# (intGreaterThanOrEqualTo)	Yes	No/No
printDuplexMode	10027	PrintDuplexMode	SET OF PrintDuplexModeSelect (setIntDoesContain)	Yes	No/No
maximumBindingMargin	10028	INTEGER	INTEGER -- max value (intGreaterThanOrEqualTo)	Yes	No/No
printFaceUpMode	10029	PrintFaceUpMode	SET OF PrintFaceUpMode (setIntDoesContain)	Yes	No/No
printStaplingSelect	10030	PrintStaplingSelect	SET OF PrintStaplingSelec(setIntDoesContain)	Yes	No/No
printPriority	10040	SimpleJobPriority	SET OF SimpleJobPriority (setIntDoesContain)	Yes	No/Yes
modeOfDataTransfer[7]	10041	DataTransferMode	SET OF DataTransferMode (setIntDoesContain)	Yes	No/No
dataLocationScheme	10042	N/A	SET OF DataLocationScheme (setIntDoesContain)	No	No/No
dataTransferTimeOutSettable	10043	N/A	BOOLEAN (boolEqualTo)	No	No/No
dataTransferTimeOutLength -- length in seconds for the FU -- to wait for the next message -- during a data transfer -- message sequence before -- detecting time-out exception	10044	INTEGER (N/A, if the previous dataTransferTimeOutSettable attribute is FALSE) -- Global attribute indicates the -- default length. If the global -- attribute value is zero, the -- default length is not fixed or -- unknown. -- If the private attribute value is -- set to zero, the FU should -- wait as long as possible. -- However, use of zero should -- be avoided.	INTEGER (intGreaterThanOrEqualTo) -- if 0, not fixed or unknown -- (use of 0 should be avoided)	Yes (No, if the previous attribute is FALSE)	Yes/No (No, if the previous attribute is FALSE)

[7] When "spoolStorage" = FALSE, only "delayed" mode is allowed for this attribute.

9.4.1.2.Dynamic Status Parameter

Dynamic Status Parameter	Query	Event	ID	Description
PrinterOperationStatus	Yes	Yes	10000	status of printing equipment.
PrinterErrorDetail	Yes	No	10001	detail error information of equipment's.
FreeStorageSize	Yes	No	10002	available storage size.
PrinterPaperInputTray	Yes	No	10003	status of paper size and direction in each input tray.
ListExcerptPrintJob	Yes	Yes	10004	lists a excerpt from print job descriptions

9.4.2.[DOC Storage] Functional Unit

9.4.2.1.Capability and Command Attribute

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private Attribute
personalityProtocol	11000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No
supportedCommand	11001	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No
dynamicStatusId	11002	N/A	SET OF DynamicStatusID (setIntDoesContain)	No	No
readWriteCapability	11003	AccessMode	SET OF AccessMode (setIntDoesContain)	No	No
minimumCheckInterval -- the minimum allowed -- value to be set in the -- checkInterval parameter -- of a SubscribeEvent -- command	11004	N/A	INTEGER (intGreaterThanOrEqualTo)	No	No
typeOfContent	11009	DataContent	SET OF DataContent (setIntDoesContain)	No	No
modeOfStore	11010	DataStoreMode	SET OF DataStoreMode (setIntDoesContain)	Yes	Yes
documentFormat	11011	DataFormat	SET OF DataFormat (setIntDoesContain)	No	No
imageCompAlgorithm	11012	ImageCompAlgorithm	SET OF ImageCompAlgorithm (setIntDoesContain)	No	No
imageByteFillOrder	11013	ByteFillOrder	SET OF ByteFillOrder (setIntDoesContain)	No	No
imageResolution	11014	ImageResolution	SET OF ImageResolution (setIntDoesContain)	No	No
dataLocationScheme	11030	N/A	SET OF DataLocationScheme (setIntDoesContain)	No	No
dataTransferTimeOutSettable	11031	N/A	BOOLEAN (boolEqualTo)	No	No

dataTransferTimeOutLength -- length in seconds for the FU -- to wait for the next message -- during a data transfer -- message sequence before -- detecting time-out exception	11032	INTEGER (N/A, if the previous dataTransferTimeOutSettable attribute is FALSE) -- Global attribute indicates the -- default length. If the global -- attribute value is zero, the -- default length is not fixed or -- unknown. -- If the private attribute value is -- set to zero, the FU should -- wait as long as possible. -- However, use of zero should -- be avoided.	INTEGER (intGreaterThanOrEqualTo) -- if 0, not fixed or unknown -- (use of 0 should be avoided)	Yes (No, if the previous attribute is FALSE)	Yes (No, if the previous attribute is FALSE)

9.4.2.2.Dynamic Status Parameter

Dynamic Status Parameter	Query	Event	ID	Description
FreeStorageSize	Yes	No	11000	available storage size.
OperatorIntervention	No	Yes	11001	a warning message to operator or administrator to request human intervention
OperatorInformation	No	Yes	11002	an informational message to operator or administrator

9.4.3.[FAX Data Send] Functional Unit

9.4.3.1.Capability and Command Attribute

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private/ Job Attribute
personalityProtocol	12000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No/No
supportedCommand	12001	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No/No
dynamicStatusId	12002	N/A	SET OF DynamicStatusID (setIntDoesContain)	No	No/No
numOfCalledSubscribers	12003	N/A	NumOfCalledSubscribers -- max integer value (intGreaterThanOrEqualTo)	No	No/No
spoolStorage	12004	N/A	SpoolStorage (boolEqualTo)	No	No/No
faxSendOrdering	12005	N/A (TelephoneNumberString be always specified when used)	FaxSendOrdering (boolEqualTo)	No	No/No

minimumCheckInterval -- the minimum allowed -- value to be set in the -- checkInterval parameter -- of a SubscribeEvent -- command	12006	N/A	INTEGER (intGreaterThanOrEqualTo)	No	No/No
documentFormat	12010	DataFormat	SET OF DataFormat (setIntDoesContain)	No	No/No
imageCompAlgorithm	12011	ImageCompAlgorithm	SET OF ImageCompAlgorithm (setIntDoesContain)	No	No/No
imageByteFillOrder	12012	ByteFillOrder	SET OF ByteFillOrder (setIntDoesContain)	No	No/No
imageResolution	12013	ImageResolution	SET OF ImageResolution (setIntDoesContain)	No	No/No
coverSheetGen	12020	CoverSheetGen	CoverSheetGen (boolEqualTo)	Yes	No/No
pageHeaderGen	12021	PageHeaderGen	PageHeaderGen (boolEqualTo)	Yes	No/No
faxProtocol	12030	FAXProtocol	SET OF FAXProtocol (setIntDoesContain)	Yes	No/No
requestPriority	12031	SimpleJobPriority (normal)	SET OF SimpleJobPriority (setIntDoesContain)	Yes	No/Yes
retryCount	12032	INTEGER	INTEGER (intGreaterThanOrEqualTo)	Yes	No/Yes
modeOfDataTransfer[8]	12035	DataTransferMode	SET OF DataTransferMode (setIntDoesContain)	Yes	No/No
dataLocationScheme	12036	N/A	SET OF DataLocationScheme (setIntDoesContain)	No	No/No
dataTransferTimeOutSettable	12037	N/A	BOOLEAN (boolEqualTo)	No	No/No
dataTransferTimeOutLength -- length in seconds for the FU -- to wait for the next message -- during a data transfer -- message sequence before -- detecting time-out exception	12038	INTEGER (N/A, if the previous dataTransferTimeOutSettable attribute is FALSE) —Global attribute indicates the —default length. If the global —attribute value is zero, the —default length is not fixed or —unknown. -- If the private attribute value is —set to zero, the FU should —wait as long as possible. —However, use of zero should —be avoided.	INTEGER (intGreaterThanOrEqualTo) -- if 0, not fixed or unknown -- (use of 0 should be avoided)	Yes (No, if the previous attribute is FALSE)	Yes/No (No, if the previous attribute is FALSE)

[8] When “spoolStorage” = FALSE, only “delayed” mode is allowed for this attribute.

9.4.3.2.Dynamic Status Parameter

Dynamic Status Parameter	Query	Event	ID	Description
FaxSendStatus	Yes	Yes	12000	status of FAX equipment at sending side.
FaxSendFreeStorageSize	Yes	No	12001	storage size available for spool.
FaxSendErrorStatus	Yes	No	12002	the detail error status information.

9.4.4.[Fax Data] Functional Unit

Refer to Part-2 Addendum.

9.5.Voice Message Systems

9.5.1.[Voice Message Storage] Functional Unit

9.5.1.1.Capability Attribute

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private/ Job Attribute
personalityProtocol	20000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No/No
supportLevel	20001	N/A	INTEGER -value should be ′one′ for Subset of [Voice Message Storage] FU (intGreaterThanOrEqualTo)	Yes	No/No
supportedCommand	20002	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No/No
dynamicStatusId	20003	N/A	SET OF DynamicStatusID (setIntDoesContain)	No	No/No
maxDuration	20020	INTEGER	INTEGER - max value (intGreaterThanOrEqualTo)	Yes	No/No
maxReceiversPlay	20021	Receiver	INTEGER - max number of receivers (intGreaterThanOrEqualTo)	Yes	No/No
maxRecipientsSend	20022	Recipient	INTEGER - max number of recipients (intGreaterThanOrEqualTo)	Yes	No/No
voiceSpeed	20023	INTEGER	BOOLEAN (boolEqualTo)	No	No/No
voiceVolume	20024	INTEGER	BOOLEAN (boolEqualTo)	No	No/No
deliveryGrade	20025	DeliveryGrade	SET OF DeliveryGrade (setIntDoesContain)	Yes	No/No
priorityLevel	20030	PriorityLevel	SET OF PriorityLevel (setIntDoesContain)	Yes	No/Yes

copyRecipients	20040	Recipient	BOOLEAN (boolEqualTo)	No	No/No
blindCopyRecipients	20041	Recipient	BOOLEAN (boolEqualTo)	No	No/No
deferredDeliveryTime	20042	UTCTime	BOOLEAN (boolEqualTo)	No	No/No
subject	20043	DisplayString	BOOLEAN (boolEqualTo)	No	No/No
maxSubjectLength	20044	N/A	INTEGER- max length of subject (intGreaterThanOrEqualTo)	Yes	No/No
synthesize	20050	N/A	BOOLEAN (boolEqualTo)	No	No/No
synthesizeVoiceSpeed	20051	INTEGER	BOOLEAN (boolEqualTo)	No	No/No
synthesizeVoiceVolume	20052	INTEGER	BOOLEAN (boolEqualTo)	No	No/No
synthesizeVoiceType	20053	INTEGER	SET OF VoiceType (setIntDoesContain)	Yes	No/No
synthesizeTextLanguage	20054	TextLanguage	SET OF TextLanguage (setIntDoesContain)	Yes	No/No
encoding	20060	Encoding	SET OF Encoding (setIntDoesContain)	Yes	No/No
minimumCheckInterval -- the minimum allowed -- value to be set in the -- checkInterval parameter -- of a SubscribeEvent -- command	20070	N/A	INTEGER (intGreaterThanOrEqualTo)	No	No/No

NOTE: [Voice Message Storage] FU defines a standard range (0 to 10, with 0 being the lowest and 10 being the highest) for voiceSpeed, voiceVolume and synthesizeVoiceSpeed. A user can specify any value in this range for the parameters corresponding to these attributes in [Voice Message Storage] FU commands.

9.5.1.2.Dynamic Status Parameter

Dynamic Status Parameter	Query	Event	ID	Description
PlayVMStatus	Yes	Yes	20000	Status of play voice message

9.6.Personal Information Systems

9.6.1.[Address Book] Functional Unit

9.6.1.1.Capability Attribute

Attribute Name	ID	Data Type as Command Attribute	Data Type as Capability Attribute (Compare Function ID)	Global Attribute	Private Attribute
personalityProtocol	30000	N/A	SET OF PersonalityProtocol (setIntIntersect)	No	No
supportedCommand	30001	N/A	SET OF SupportedCommand (setIntDoesContain)	No	No
exchangeDataFormatSupport	30010	ExchangeDataFormat	Set OF ExchangeDataFormat (setIntDoesContain)	No	No
characterSetSupport	30011	CharSetID	SET OF CharSetID (setIntDoesContain)	No	No
searchSupport	30012	N/A	SearchSupport (boolEqualTo)	No	No
sortSupport	30013	N/A	SortSupport (boolEqualTo)	No	No

10.Basic Encoding Rule (BER)

The encoding of protocol data unit follows the "specification of basic encoding rules for abstract syntax notation one (ASN.1)" as defined by ISO 8825.

11.References

- ISO 8824 Information processing systems - Open systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)
- ISO 8825 Information processing systems - Open systems Interconnection - Specification of basic encoding rules for Abstract Syntax Notation One (ASN.1)
- Sun Microsystems, “RPC: Remote Procedure Call Protocol Specification Version 2”, RFC-1057, June 1988
- Rose & McCloghrie, “Structure and Identification of Management Information for TCP/IP-based Internets”, RFC-1155, May 1990
- Case, Fedor, Schoffstall, & Davin, “A Simple Network Management Protocol (SNMP)”, RFC-1157, May 1990
- SNMP Working Group, “Management Information Base for Network Management of TCP/IP-based internets: MIB-II”, RFC-1213, March 1991
- Berners-Lee, Masinter & McCahill, “Uniform Resource Locators (URL)”, RFC-1738, December 1994
- versit Consortium, “vCard The Electronic Business Card Version 2.1”, September 18,1996