DIGITAL MEDIA BRIDGE SENDER USER'S GUIDE

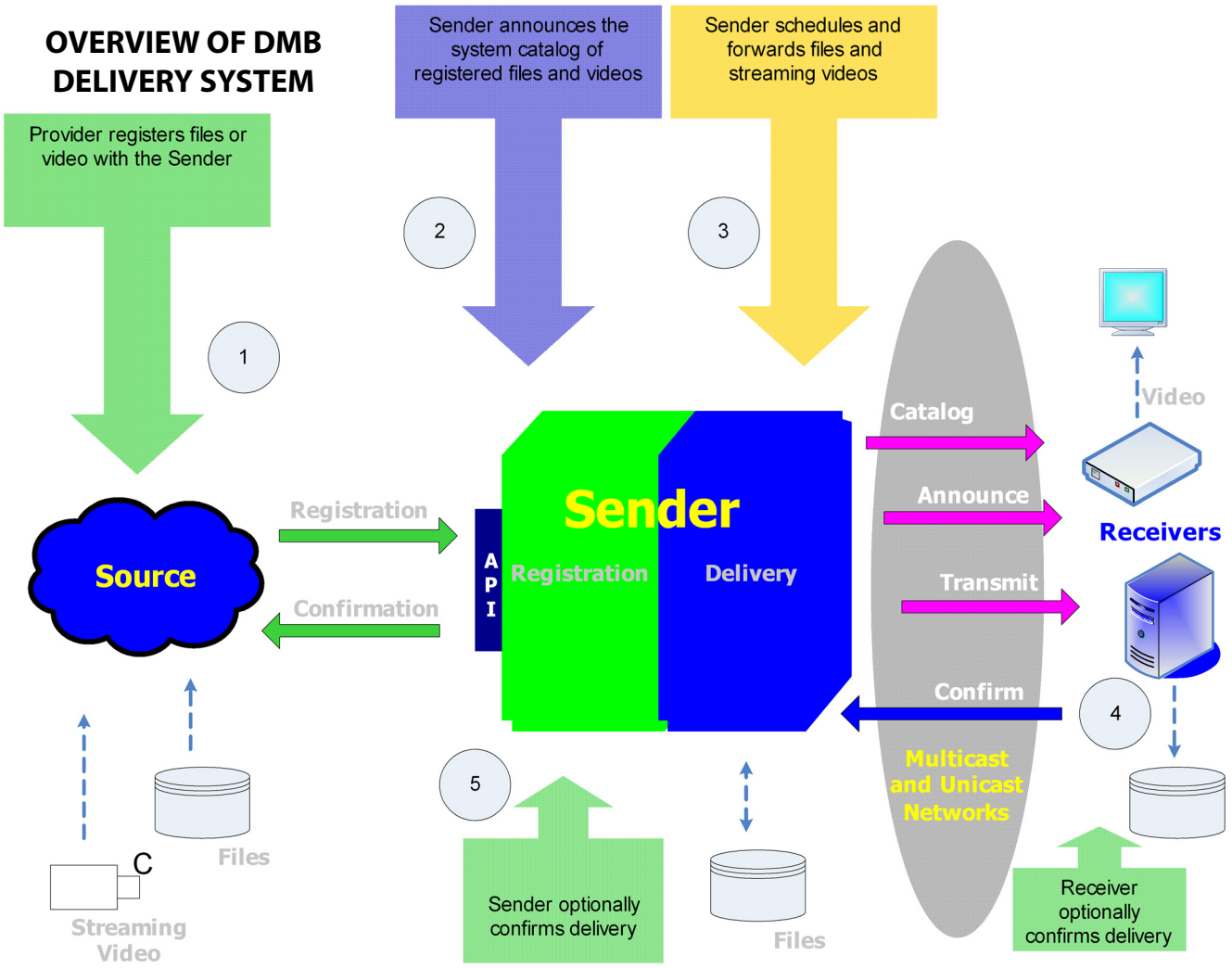
Cilutions - Digital Media Bridge
877-515-4004

www.cilutions.com

Contents

Digital Media Bridge (DMB) is a content and video distribution family of products useful for distributing files and video streams from a sending machine to multiple IP-based multicast and unicast receiving machines in a connected network. It operates in both multicast-enabled enterprise networks and unicast-only networks, both private and public.

The following diagram shows the end-to-end flow of files and video from a customer's source location through the DMB platforms to the delivery end point. The Sender platform accepts source content and delivery commands from the user, manages the outbound multicast groups, manages unicast destinations, initiates file transfers, initiates streaming video relay and controls and meters the transmission process. The Receiver platform joins multicast groups, listens on unicast ports, receives and processes files and streaming video sent by the Sender, and provides reception status to the Sender when requested.



DMB supports distribution to both "receive-only" sites (i.e., with no back channel to the Sender) and two-way sites. DMB is especially tolerant of networks where connectivity to one or more destinations is intermittent and can guarantee delivery to end points operating in such an environment.

DMB also supports file retrieval capability where the Sender can pull a file from a Receiver or group of Receivers back to a central server for processing. In this way DMB supports distribution both outbound and inbound.

This document describes the operational aspects for Users of the DMB system. It concentrates on how a user interacts with the DMB Sender in submitting files and video to the network and controls their delivery.

Introduction : Types of Sender User Interfaces

The DMB Sender supports interfaces for both content (i.e., the files or streaming video source) and command operations (i.e., the instructions to the Sender controlling the transmission of the content). As depicted in the diagram below the user provides content one of two ways. Files, either video or data, arrive through an FTP connection from the user to the Sender. The files are placed either on the Sender's local hard drive or on a network shared drive accessible to the Sender. Real-time streaming video originates at the time of transmission (say from an encoder with multicast connectivity to the Sender) and the Sender relays the v ideo stream from the source to the destination receivers in real-time; the Sender does not store real-time streaming video on disk.

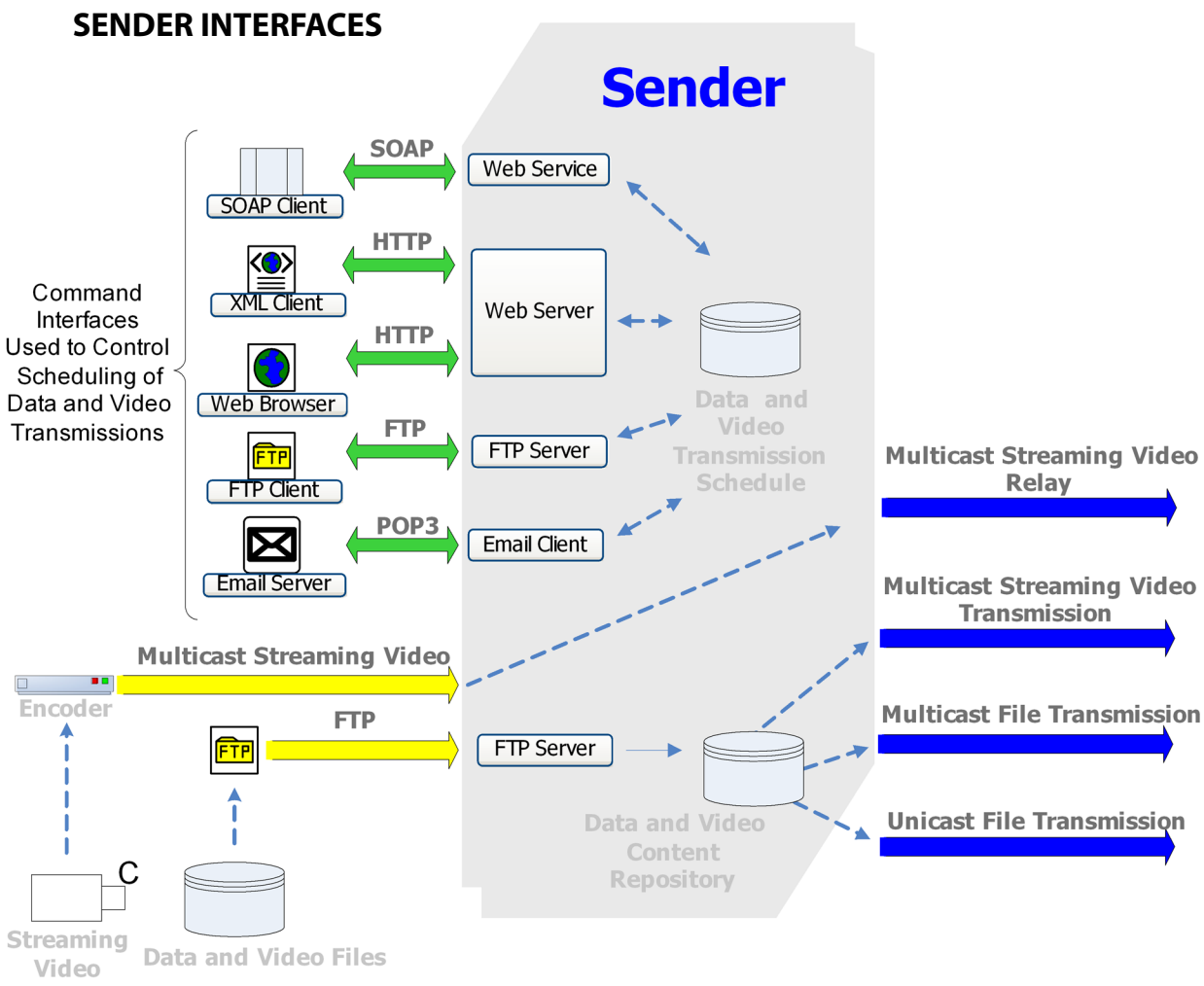
After files have been deposited on the Sender the user issues transmission related commands using one more of the following interfaces:

• FTP – The user builds XML-formatted (described elsewhere in this document) or plain text command files and places them in a well-known directory on the Sender. Command results are generated by the Sender and placed in a well-known results directory for the user's review.

• Browser User Interface – Both HTML and Flash-based Web Portals on the Sender offer user control of transmissions and monitoring of results.

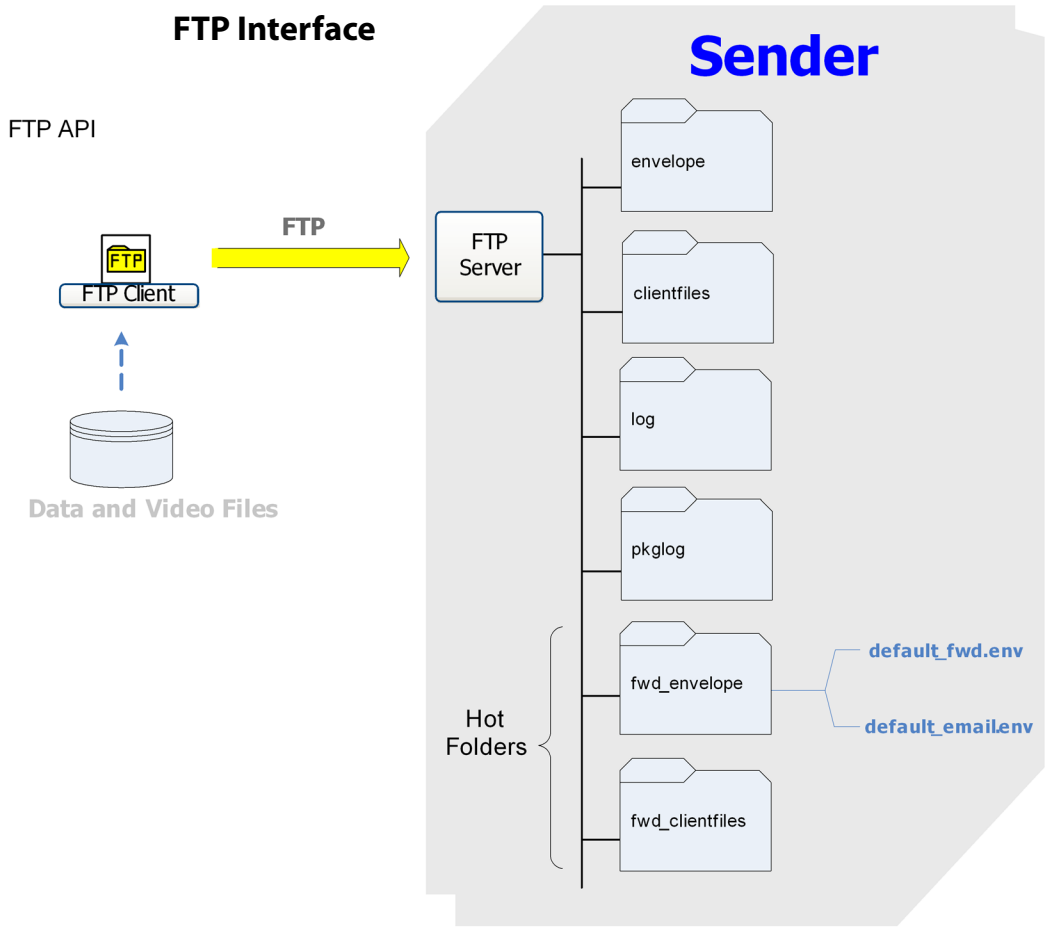
• XML Server to Server Interface – XML Command/Response offers programmatic control of transmissions and results monitoring.

A Software Development Kit (SDK) is available with example programs demonstrating how to interact with the Sender using the multiple APIs supported.



FTP Interface

The Sender administrator sets up and enables one or more FTP accounts on the Sender, one for each user login, offering access as depicted in the following diagram:



The root directory offers the following subdirectories used as an interface between the user and the Sender:

• **.../clientfiles** - The video or data file that the Sender transmits to one or more Receivers. The user puts files here before registering them using one of the command interfaces.

• **.../envelope** – File transmission commands controlling the registration and transmission attributes of files in the .../clientfiles directory. These *commands* are text or XML formatted files which serve as instructions to the Sender. The types of files are:

- *XML* - Identified as files with an .xml extension and containing XML formatted commands. The precise command format is described in the *Command XML Format* section of this document.
- *Plain Text* - Identified as files without an .xml or .vfx extension and containing plain text formatted commands. The complete set of pain text attributes and the format of this file can be viewed in the Web Forms interface using the "Render" button on the New Package screen. Context sensitive help is also provided there as a helpful guide.
- *VFX* - Indentified as files with a .vfx extension and containing plain text registration and control information for multicast streaming video relay transmissions. The specific format of this file and how it controls video registrations are presented in the diagram on Video Relay in *Appendix B*.

• **.../log** - Sender results of commands processed from the .../envelope directory. When the Sender detects a file in the .../envelope directory it performs the requested operation (e.g., registers a new file for transmission), removes the file from the .../envelope directory and puts the results of the attempted operation in the .../log directory. The name of the log file is the same name of the file in the .../envelope directory but with _log appended. If the original command in the .../envelope directory was XML formatted (i.e., file name ended in .xml) then the .../log file is an XML response; otherwise the .../log file is a plain text description of the results.

• **.../pkglog**- User specific daily logging as described in *Appendix C*.

• **Hot Folders** - Designed to offer "1 step registration" of video or data files these directories let a user send content by simply FTPing the file into a known directory. The registration commands are set up ahead of time so the user does not need to explicitly supply a command for each new file. The directories involved are:

- .../*fwd_clientfiles* - The Hot Folder Forward directory. When the Sender finds a file here it automatically moves it into the .../clientfiles directory and executes the default Hot Folder Registration command placing the results of the command in the .../log directory.
- .../*fwd_envelope* - The Hot Folder Registration directory. This directory consists of two files which can be modified by a user to set the default command profile for Hot Folder video and data files. These are:
  - default_fwd.env - The command to run against any and all files which appear in the .../fwd_clientfiles directory.
  - default_email.env - The command to run against any and all files which arrive as an attachment to an email.

An example default_fwd.env file used to automatically register files arriving in .../fwd_clientfiles and send them to sites in the group named MYGROUP is as follows:

```
#Basic Envelope which removes any existing
#registration for a file then registers it for distribution
#to all sites in group TESTGROUP.  The file will use the
#default attributes configured for this client (e.g., for
#transmission rate, delivery assurance). To override
#these default values include any new attributes in the
#body of the second envelope below. Use the "Render"
#button in the New Package screen on the Web Forms
#for help on these fields Note that the FILE_NAME and
#DEST_FILE_NAME attributes may not be modified.

#Registration results for each file placed in .../fwd_client
#files can be found in the .../log directory. The log file
#name is in <filename>_log format.
BEGIN
FILE_NAME       <<FILENAME>>
DEST_FILE_NAME   <<FILENAME>>
DESTINATION
BEGIN
END
END

BEGIN
FILE_NAME     <<FILENAME>>
DEST_FILE_NAME <<FILENAME>>
DESTINATION
BEGIN
@MYGROUP, default
END
END
```

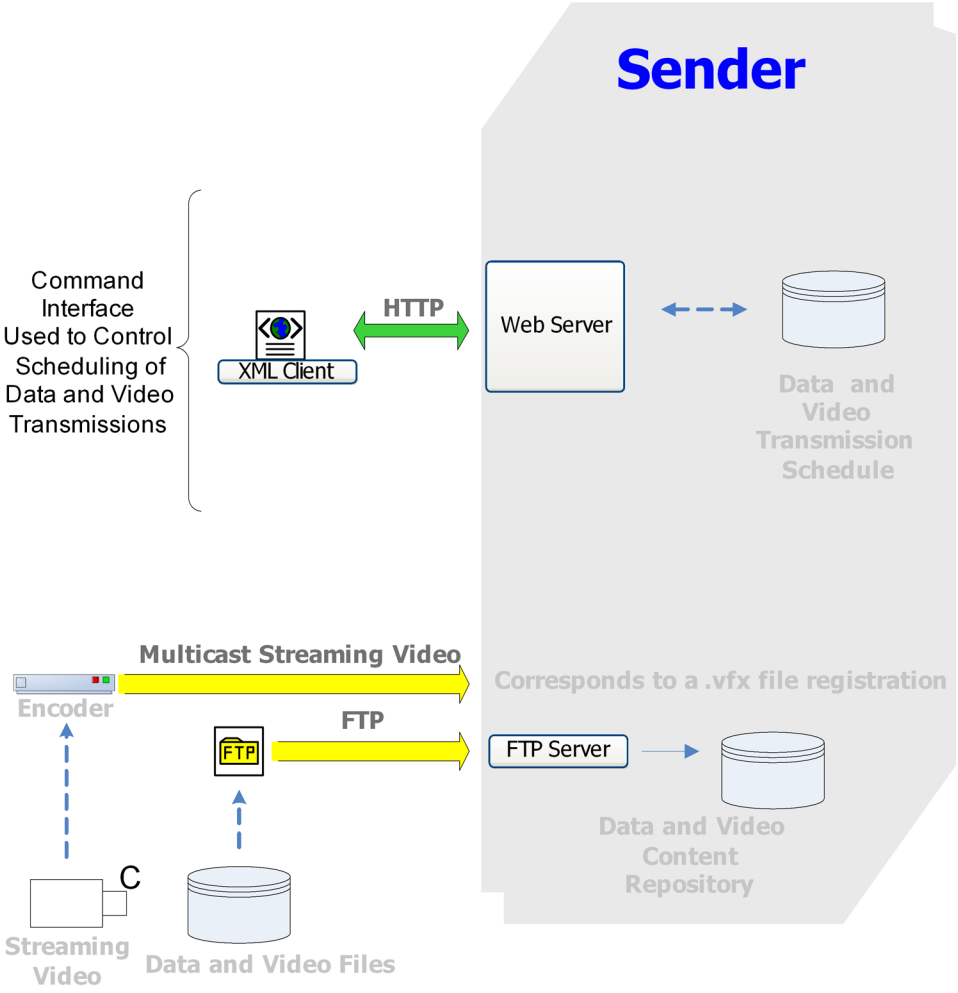
An example default_email.env file used to automatically register files arriving as email attachments and send them to sites in the group named MYGROUP is as follows:

```
#DESC_NAME=email subject
#TOPIC=email subject
#DESCRIPTION=email body
#FILENAME=email attachment
BEGIN
FILE_NAME      <<FILENAME>>
DEST_FILE_NAME  <<FILENAME>>
DESTINATION
BEGIN
END
END

BEGIN
FILE_NAME      <<FILENAME>>
DEST_FILE_NAME  <<FILENAME>>
ASSURANCE      CONFIRM
DESC_NAME      <<DESC_NAME>>
TOPIC          <<TOPIC>>
DESCRIPTION    <<DESCRIPTION>>
DESTINATION
BEGIN
@MYGROUP, default
END
END
```

Web Browser Interface

The Sender administrator will set up and enable Web Browser access to the Sender offering the following interface:

SENDER WEB BROWSER INTERFACE



Here a user typically posts a video or data file to the Sender using FTP access then, using a Web Browser (e.g., Internet Explorer), opens the URL provided to the user by the Sender Administrator. There are two Web Browser URLs offering two types of access:

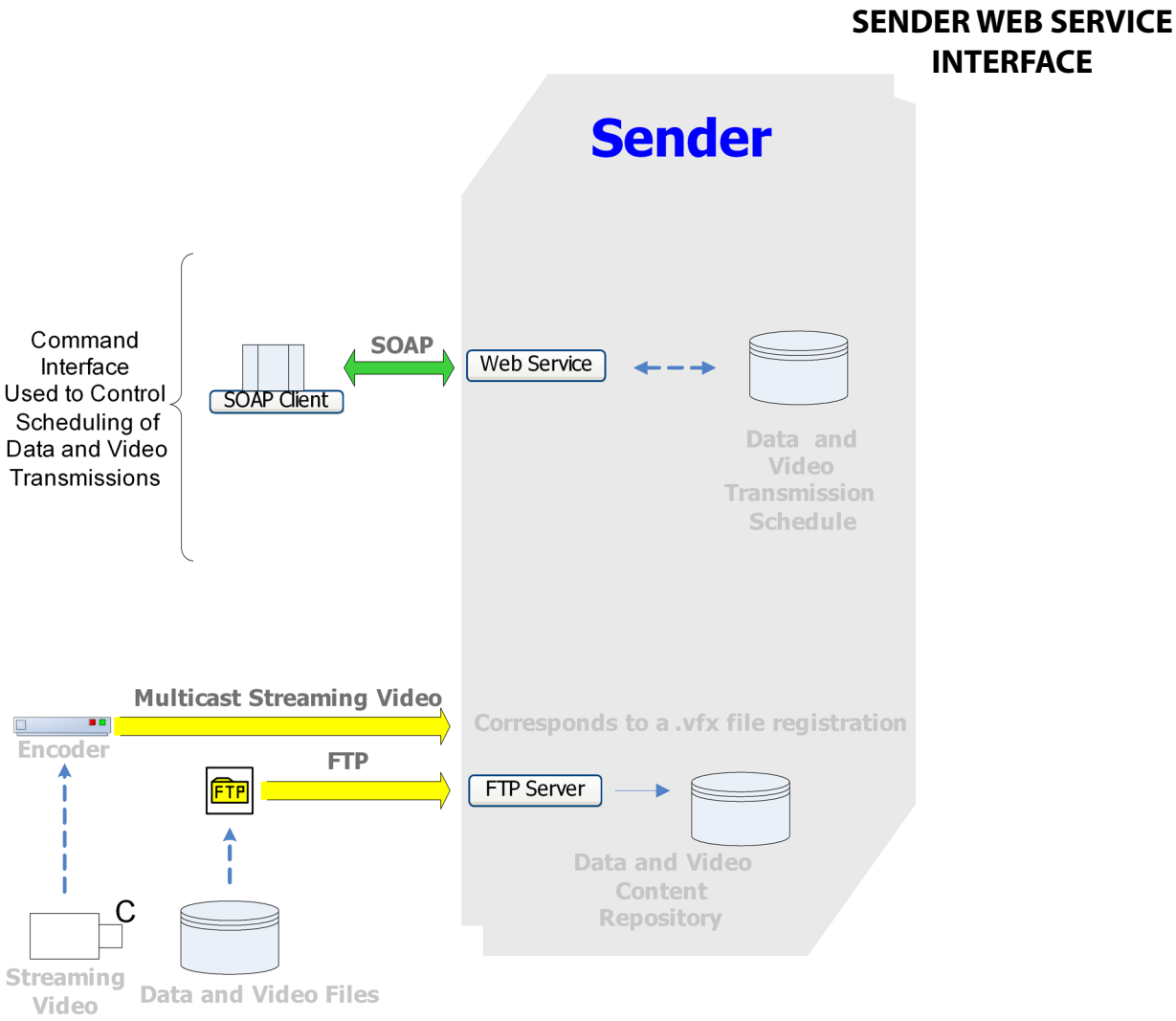
• **Flash Forms** - a Rich Internet Application (RIA) web portal requiring a Flash plug-in within the user's browser. These forms offer:

- New File Registration
- File Status Monitoring
- Group Management
- Bandwidth Controls
- Optional Digital Signage Screen Management (*works with Cilutions' provided remote media players*)

• **Classic Forms** - Internet Explorer access providing all the capabilities of the Flash Forms in a classic view.

Web Service Interface

The Sender administrator will set up and enable Web Service access to the Sender offering the following interface:



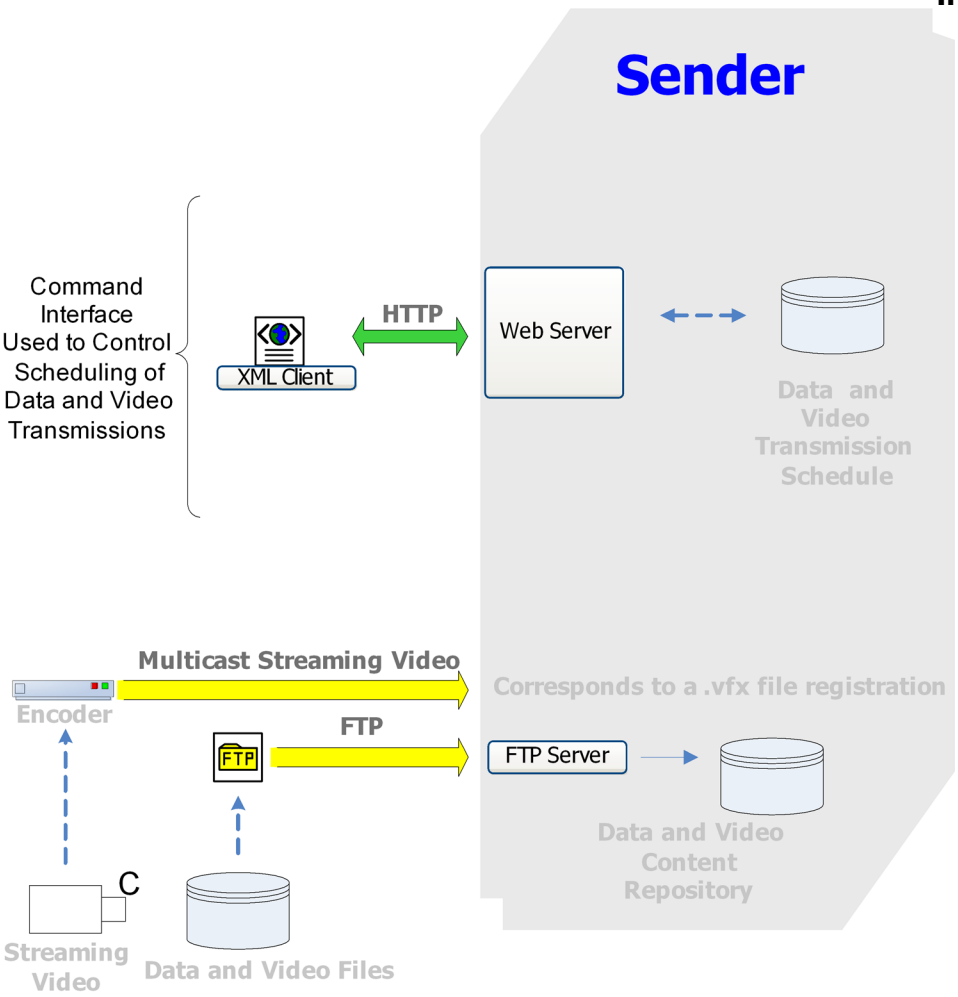
Here a user typically posts a video or data file to the Sender using FTP access then programmatically controls file and video registration and management. The SOAP client issues commands to the Sender SOAP Web Service and receives responses. The formal command set is provided separately as part of the Sender Software Development Kit (SDK) which includes sample programs demonstrating how a SOAP client can interact with the Sender.

The command set broadly corresponds to the list of commands and responses for the XML Command/Response interface described in detail later in this document.

XML Client Interface

The Sender administrator will set up and enable XML Client access to the Sender offering the following interface:

SENDER XML CLIENT INTERFACE



Here a user typically posts a video or data file to the Sender using FTP access then programmatically controls file and video registration and management. The XML client issues commands to the Sender Web Server and receives responses. The formal command set is provided separately as part of the Sender Software Development Kit (SDK) which includes sample programs demonstrating how an XML client can interact with the Sender.

This section provides a description of the individual Sender transactions. The examples shown demonstrate actual transactions in XML format (as files or as http command response). These transactions are also supported as plain text files and in SOAP command/ responses format.

DMB Transactions : Parameter Attributes

The classes of transaction parameters are as follows:

- Required – these must be supplied at the time of the transaction. They are rendered in **bold**.
- Profiled – these are mandatory parameters which can be preset in a Provider's Profile to be used as a default if they are not provided at the time of the transaction. They are rendered in *italics*.
- Optional – these are not required. The Sender will use built-in defaults if they are not provided at the time of the transaction. They are rendered in regular font with the built-in default provided in the parameter's description.

The attributes of commonly used transaction parameters are as follows:

- GROUP – An up to 18 character, upper case, alpha-numeric string with no spaces.
- SOURCE_CLIENT – An up to 10 character, upper case, alpha-numeric string with no spaces.
- SiteId – An 8 character, upper case, alpha-numeric string with no spaces.
- FILE_NAME – An up to 254 character alpha-numeric string. Spaces are permitted.

DMB Transactions : XML Transaction Model

This interface supports posting XML formatted commands to the Sender and receiving XML formatted responses. These commands are either files (identified by a .xml extension) placed in the …*/envelope* directory for this user or within the body of an HTTP POST to the Sender CGI program.

DMB Transactions : XML interface Components

- XML Commands – – issued by the client application. A sample html form for posting XML commands is as follows:

```
<HTML>
  <HEAD>
    <TITLE>SFXParse ISAPI Extension</TITLE>
  </HEAD>
  <BODY bgcolor="white" topmargin="10" leftmargin="10">
    <h2>SFXParse ISAPI Extension</h2>
     <p>Place XML envelope in the input text area and press "Submit"</p>
     <form action="http://localhost/pdxml/sfxparse_iis.exe"
method="POST">
     <textarea name=xml cols=80 rows=20 wrap=OFF></tex¬tarea>
     <br>
     <input type="submit" value="Submit">
    </form>
  </BODY>
</HTML>
```

This form can be rendered in a Web browser. XML commands, described below, can be pasted into the textarea of this form and submitted to the Sender.

- XML Responses – Returned by the Sender in response to an XML command. The response is command specific as described below for each command.
- DTD– The Document Type Definitions used by the XML Parser to analyze the commands and responses. These are stored on the Sender and are accessible as a URL reference.

DMB Transactions : XML Security

Each user can be assigned (by the Sender Administrator) a user account password. If configured, each transaction from the corresponding user must contain the password as a key to authenticate the user. Here is a sample XML command which requests a summary of packages currently registered by the client named ADMIN. The command includes a key of abc123.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE packageSummaryCmd SYSTEM "../dtd/pd/1.0/
pack¬ageSummaryCmd.dtd">
<packageSummaryCmd>
        <sourceClient name="ADMIN" key="abc123"/>
</packageSummaryCmd>
```

The connection from the user to the Sender should be over a secure, private, network connection. If using the Internet this should be a Virtual Private Network (VPN).

This section defines the individual commands and responses supported by the Sender XML Interface.

Each command is identified and its parameters are described. An XML sample is also provided for each command. There is a plain text description of each response and its corresponding DTD is referenced where the precise XML format can be derived.

The response to many of the commands is pass/fail. Failures also contain an error code and text string. The complete list of codes and strings is provided in *Appendix A*.

A sample XML formatted pass response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response SYSTEM ".../dtd/pd/1.0/response.dtd">
<response>
<pass/>
</response>
```

A sample XML formatted failure, with error code and descriptive message, is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response SYSTEM ".../dtd/pd/1.0/response.dtd">
<response>
<fail error="7">stop_time is in the past Tue May 29 12:32:00 2009
</fail>
</response>
```

Commands and Responses : **Group**

1. *Command* - There is a single command for this transaction, named GROUP. The DMB Sender creates a new group, redefines an existing group, or deletes an existing group depending on the contents of the DESTINATION parameter.

2. *Parameters* - The following table describes the specific parameters within the GROUP transaction.

Entry	Description
Group	The unique name of the group.
SOURCE_CLIENT	The source client name of an Information Provider with an active account.
DESTINATION	The destination field consists of a list of Receiver SiteIds. The format for the DESTINATION field entries is one Receiver SiteId per line. An empty list (no siteid entries) denotes that the group is to be deleted.

3. *Command XML Format* - The group command DTD is groupCmd.dtd. A sample command is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE group SYSTEM ".../dtd/pd/1.0/groupCmd.dtd">
<group name="GROUP1">
        <sourceClient name="ADMIN"/>
        <destination>
                <site id="TSTSITE1"/>
                <site id="TSTSITE2"/>
        </destination>
</group>
```

4. *Response* - The group response DTD is response.dtd and contains:

Entry	Description
STATUS	Indicates: • PASS • FAIL
ERROR	For a failed transaction, indicates: <errno>

Commands and Responses : **Group Status**

1. *Command* - There is a single command for this transaction, named GROUP_STATUS. The Sender queries its local DB and returns the group's status as a response.

2. *Parameters* - The following table describes the specific parameters within the GROUP_STATUS transaction.

Entry	Description
GROUP	The name of the group
SOURCE_CLIENT	The source client name of an Information Provider with an active account

3. *Command XML Format* - The groupStatus command DTD is groupStatusCmd.dtd. A sample command is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE groupStatusCmd SYSTEM ".../dtd/pd/1.0/groupStatusCmd.dtd">
<groupStatusCmd>
        <groupInfo name="GROUP1"/>
        <sourceClient name="ADMIN"/>
</groupStatusCmd>
```

4. *Response* - The groupStatusResponse response DTD is groupStatusResponse.dtd and contains:

Entry	Description
STATUS	Indicates: • ACTIVE • FAIL
ERROR	For a failed transaction, indicates: <errno>
DESTINATION	The destination field consists of a list of Receiver SiteIds, one per line

Commands and Responses : **Group Summary**

1. *Command* - There is a single command for this transaction, named GROUP_SUMMARY. The Sender returns the complete list of groups currently configured by this client.

2. *Parameters* - The following table describes the specific parameters within the GROUP_SUMMARY transaction.

Entry	Description
SOURCE_CLIENT	The source client

3. *Command XML Format* - The groupSummary command DTD is groupSummaryCmd.dtd. A sample command is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE groupSummaryCmd SYSTEM ".../dtd/pd/1.0/groupSummaryCmd.dtd">
<groupSummaryCmd>
        <sourceClient name="ADMIN"/>
</groupSummaryCmd>
```

4. *Response* - The groupSummaryResponse response DTD is groupSummaryResponse.dtd and contains:

Entry	Description
STATUS	Indicates: • PASS – SOURCE_CLIENT valid • FAIL – SOURCE_CLIENT invalid
ERROR	For a failed transaction, indicates: <errno>
DESTINATION	The groups field consists of a list of groups for this client, one per line.

A sample group summary response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE groupSummaryResponse SYSTEM ".../dtd/pd/1.0/groupSummaryResponse.dtd">
<groupSummaryResponse>
<groups>
<groupInfo name="TEST"/>
<groupInfo name="TEST2"/>
<groupInfo name="TEST1"/>
</groups>
</groupSummaryResponse>
```

1. *Command* - There is a single command for this transaction, named PACKAGE. The Sender creates a new package, modifies the attributes of an existing pack age, or unregisters an existing package depending on the context of the referenced package or the contents of the parameters. If the package is already registered at the Sender then its attributes (e.g., its destination list) may be modified by the new parameter settings. If the DESTINATION list is empty then the package will be unregistered.

2. *Parameters* - The following table describes the specific parameters within the PACKAGE transaction

Entry	Description
SOURCE_CLIENT	The source client name informs the Sender the client requesting to broadcast a package and the directory where the package is located on the Sender.
FILE_NAME	The up to 255 alphanumeric character filename of the package on the Sender. Ths name can contain spaces.
DEST_FILE_NAME	The up to 255 alphanumeric character filename to use when delivering the package to the Receiver. This field lets a client specify a destination package name that is different from the name on the Sender. If this field is missing then it uses the same name as the package on the Sender.
ASSURANCE <param>	Indicates what measures are to be taken to ensure the package is delivered to all addressed receivers. Its options are: 1) RETRANSMIT where remotes request retransmission of only missed portions of a package, if any; 2) CONFIRM which includes RETRANSMIT and causes each receiving remote to explicitly notify the Sender when it has successfully received the package; and 3) BEST_EFFORT N where N is a single decimal digit indicating how many times to transmit the package in its entirety. It can be from 1 to 3. In this last case there is no remote initiated retransmission or confirmation.
INTERACTIVITY <param>	Designates the initial transmission method. Its options are: 1) PUSH to send the package assuming each remote is listening and ready to receive it; 2) PULL to wait for at least one remote to request it before sending the package. A remote will request such a package automatically without requiring operator action; and 3) REQUEST which is the same as PULL except a remote operator must explicitly request the package
BIT_RATE <param>	The bit rate (Kbits/s) at which the Sender should transmit the package.
PRIORITY <param>	The relative transmission priority applied to the package. This determines the order in which packages will be transmitted and preempted by the Sender when several packages are scheduled for transmission at the same time. There are 8 priority levels, with 1 the highest, and 8 the lowest.

Entry	Description
START_TIME <param>	Indicates the earliest time, in YYYY/MM/DD HH:MM:SS format, the package may begin transmitting. The default is now.
STOP_TIME <param>	Indicates the latest time, in YYYY/MM/DD HH:MM:SS format, the package may begin transmitting. The default is set in the client's profile as the number of seconds after the START_TIME.
DESC_NAME <param>	The up to 40 alphanumeric character descriptive name, with spaces allowed, that is used by the package explorer to organize the presentation of the package to the user at the Receiver.
TOPIC <param>	The up to 10 alphanumeric character descriptive name, with no spaces, that is used by the package explorer to organize the presentation of the package to the user at the Receiver.
DESCRIPTION <param>	The up to 510 alphanumeric character descriptive name, with spaces allowed, that is used by the package explorer to organize the presentation of the package to the user at the Receiver.
STATUS_REPORTING <param>	The manner in which status (e.g., results of registration) should be returned to the client. Its options are: 1) EMAIL indicating that an email status report should be sent to the client reporting transaction status; 2) LOG indicating that the Sender should keep the transaction status in its internal log files only.
EMAIL_TEXT	The client supplied text that will be included in the package transaction status report. This text is placed in the email message immediately following the email header and prior to the status report body.

Entry	Description
EXPIRATION <param>	The type of expiration which applies to the package. Its values are Manual (client will manually remove the package registration), Success (the Sender will remove the registration when all siteids have confirmed package reception), and Auto (the Sender will automatically remove the package registration when its stop time expires).
LINK <filename>	The actual file name of the package on the Sender. The Sender created a link from this value to the Source Package Name of this transaction. In this way a client can submit multiple views of the same package without requiring additional disk space for each view. For example, this can be used to load the same image to different sets of Receivers at different times.
DELIVER_ENVELOPE <param>	*Yes* indicates the envelope (i.e., the internal Sender file showing the package's attributes) should be delivered to each Receiver creating the envelope subdirectory if necessary. *No* indicates it should not be delivered, and *Default* indicates it should be delivered only if the envelope subdirectory already exists on the Receiver. The default is *Default*.

Entry	Description
CONF_DEADLINE <param>	The confirmation deadline of the package in YYYY/MM/DD HH:MM:SS format. When this time arrives the Sender generates a confirmation report, if a Status Email Address has been provided, and emails it to the client. If the Expiration Type attribute is *Auto*, the package will be expired at this time. This field must be far enough beyond the Stop Time of the package to allow for a complete package transmission with confirmations from Receivers distributed over a random backoff. Contact the Sender Administrator to determine the maximum time a remote can wait before confirming a package.
FEC_OVERHEAD	The level of forward error correction, 0 to 99, applied to the package during transmission. This is typically used when sending content to '*receive only*' Receivers. Each Receiver, if needed, will use FEC to attempt to restore lost packets during package reception. This value determines, as a percentage of the total size of the package, the amount of error correction to apply. The Sender transmits all FEC data, if enabled, immediately following a package's transmission.
DOSIGNATURE	This is a 128-bit MD5 checksum of the package. If enabled, it is generated at the Sender and regenerated and verified at each Receiver. This provides end-to-end, package level, data integrity validation.
DO_DELETE	Remove this file from the Sender when this Package Expires.

Entry	Description
DESTINATION	The destination field identifies the Receivers for which the package is intended. It contains the list of destination SiteIds with a destination <alias> (an alias of *DEFAULT* points to the ...*load* directory on the receiver). An empty destination list removes the file registration on the Sender. The destination field can also include one of the following other two options. • BROADCST, <alias> - A single line indicating that the package is for all Receivers tuned to this Sender. • @GROUPNAME, <alias> - A single line indicating that the package is for all Receivers which are members of the designated group.

3. *Command XML Format* - The package command DTD is packageCmd.dtd. A sample command is follows. It shows the unregistration of a file named sample.txt and it reregistration in a single transaction.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE package SYSTEM ".../dtd/pd/1.0/packageCmd.dtd">
<package>
        <contentInfo>
                <sourceClient name="ADMIN"/>
                <fileName value="sample.txt"/>
                <destFileName value="sample.txt"/>
                <destinations>
                        <destination/>
                </destinations>
        </contentInfo>
        <contentInfo>
                <sourceClient name="ADMIN"/>
                <fileName value="sample.txt"/>
                <destFileName value="sample.txt"/>
                <assurance>
                        <confirm/>
                </assurance>
                <interactivity type="PUSH"/>
                <bitRateInfo>3000</bitRateInfo>
                <priorityInfo value="HIGH"/>
                <time start="2009/05/29 12:32:00"
stop="2010/05/29 12:32:00"/>
                <descName>test desc name</descName>
                <topic>test topic</topic>
                <description>test description</description>
                <community name="ALLSITES"/>
                <statusReporting email=""/>
                <emailText>Test EMail Text</emailText>
                <expiration value="MANUAL"/>
                <confDeadline>2009/05/30 12:32:00</
confDeadline>
                <deliverEnvelope value="DEFAULT"/>
                <fecOverhead>10</fecOverhead>
                <dosignature value="YES"/>
                <destinations>
                        <destination>
                                <site id="TSTSITE1"
destdir="CLIENT1"/>
                                <site id="TSTSITE2"/>
                        </destination>
                </destinations>
        </contentInfo>
</package>
```

4. *Response* - The package response DTD is response.dtd and contains:

Entry	Description
STATUS	Indicates: • PASS • FAIL
ERROR	For a failed transaction, indicates: <errno>

Commands and Responses : Package Summary

1. *Command* - There is a single command for this transaction, named PACKAGE_SUMMARY. The Sender returns the complete list of packages currently registered by this client.

2. *Parameters* - The following table describes the specific parameters within the PACKAGE_SUMMARY transaction.

Entry	Description
SOURCE_CLIENT	The source client name.

3. *Command XML Format* - The packageSummary command DTD is packageSummaryCmd.dtd. A sample command follows. It requests the list of files names, along with their file ids, for all files currently registered by client ADMIN.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE packageSummaryCmd SYSTEM ".../dtd/pd/1.0/pack-
ageSummaryCmd.dtd">
<packageSummaryCmd>
        <sourceClient name="ADMIN"/>
        <detailsLevel value="FILENAME_ID"/>
</packageSummaryCmd>
```

4. *Response* - The packageSummary response DTD is packageSummaryResponse.dtd and contains the list of file names and ids as follows:

Entry	Description
STATUS	Indicates: • PASS • FAIL
ERROR	For a failed transaction, indicates: <errno>
FILE_NAME	The file_names field consists of a list of Packages for this client. The format for the FILE_NAMES field entries is one package per line.
FILE_ID	The fileid assigned by the Sender to identify this file's current registration. This is a unique number, systemwide, that identifies the instance of a file. That is, if a file is unregistered by name then registered again the Sender will assign it a new, ever increasing, file id.

A sample package summary response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE packageSummaryResponse SYSTEM "../dtd/pd/1.0/
packageSummaryResponse.dtd">
<packageSummaryResponse>
<fileID value="27"/>
<fileName value="1k.txt"/>
<fileID value="28"/>
<fileName value="filegen.exe"/>
<fileID value="29"/>
<fileName value="sample.txt"/>
</packageSummaryResponse>
```

Commands and Responses : **Package Status**

1. *Command* - There is a single command for this transaction, named PACKAGE_STATUS. The Sender queries its local DB and returns the package's status as a response.

2. *Parameters* - The following table describes the specific parameters within the PACKAGE_STATUS transaction.

Entry	Description
SOURCE_CLIENT	The source client
FILE_NAME	The file name of the package

3. *Command XML Format* - The packageStatus command DTD is packageStatusCmd.dtd. A sample command is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE packageStatusCmd SYSTEM "../dtd/pd/1.0/packageS-
tatusCmd.dtd">
<packageStatusCmd>
        <sourceClient name="ADMIN"/>
        <filename value="filegen.exe"/>
</packageStatusCmd>
```

4. *Response* - The packageStatus response DTD is packageStatusResponse.dtd and contains:

Entry	Description
FILE_NAME	The name of the file
FILE_ID	The fileid assigned by the Sender to identify this file's current registration. This is a unique number, systemwide, that identifies the instance of a file. That is, if a file is unregistered by name then registered again the Sender will assign it a new, ever increasing, file id.
STATUS	activeStatus indicates that the stop time for this package has not yet arrived. inactiveStatus indicates that it has arrived and the package will no longer be transmitted by the Sender.
TRANSMISSIONS	Indicates the number of times all or parts of the package have been transmitted.
QUEUED	Indicates if the file is currently queued for transmission.
TRANSMITTING	Either YES or NO to indicate if the package is currently transmitting.
REGISTRATION_TIME	Time the package was registered at the Sender.
FIRST_TRANSMITTED	Time the package was first transmitted.
LAST_TRANSMITTED	Time the package was last transmitted.
DESTINATION_GROUP	Destination Group Name, if any

A sample good package status response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE packageStatusResponse SYSTEM "../dtd/pd/1.0/packageStatusResponse.dtd">
<packageStatusResponse>
<fileName value="filegen.exe"/>
<fileID value="28"/>
<activeStatus/>
<transmissions>1</transmissions>
<queued>NO</queued>
<transmitting value="NO"></transmitting>
<registrationTime>Tue Dec 17 18:02:05 2009</registrationTime>
<firstTransmitted>Tue Dec 17 18:02:13 2009</firstTransmitted>
<last_transmitted>Tue Dec 17 18:02:13 2009</last_transmitted>
<destinationGroup></destinationGroup>
</packageStatusResponse>
```

A sample failed package status reponse (for a file which is not registered in the Sender) is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE packageStatusResponse SYSTEM "../dtd/pd/1.0/packageStatusResponse.dtd">
<packageStatusResponse>
<unknownStatus/>
</packageStatusResponse>
```

Commands and Responses :
Package Confirmation Report

1. *Command* - There is a single command for this transaction, named PACKAGE_CONFIRMATION. The Sender returns a confirmation report listing the delivered status of the package at each destination site where the package was addressed.

2. *Parameters* - The following table describes the specific parameters within the PACKAGE_CONFIRMATION transaction.

Entry	Description
SOURCE_CLIENT	The source client name
FILE_NAME	The file name of the package unique to this client
FILE_ID	The fileid assigned by the Sender to identify this file's current registration. This is a unique number, systemwide, that identifies the instance of a file. That is, if a file is unregistered by name then registered again the Sender will assign it a new, ever increasing, file id

3. *Command XML Format* - The packageSummary command DTD is confirmationStatusCmd.dtd. A sample command follows. It requests a status report of the package with a fileid of 30

```
<!DOCTYPE confirmationStatusCmd SYSTEM "../dtd/pd/1.0/confirmationStatusCmd.dtd">
<confirmationStatusCmd>
        <sourceClient name="ADMIN"/>
        <fileID value="30"/>
</confirmationStatusCmd>
```

Here is a similar command requesting the report by file name instead of id:

```
<!DOCTYPE confirmationStatusCmd SYSTEM "../dtd/pd/1.0/confirmationStatusCmd.dtd">
<confirmationStatusCmd>
        <sourceClient name="ADMIN"/>
        <fileName value="4MB.txt"/>
</confirmationStatusCmd>
```

4. *Response* - The package confirmation response DTD is confirmationStatusResponse.dtd and contains the list of file names and ids as follows:

Entry	Description
STATUS	Indicates: • Pass • Fail
ERROR	For a failed transaction, indicates: <errno>
FILE_NAME	The file_names field consists of a list of packages for this client. The format for the FILE_NAMES field entries is one package per line
FILE_ID	The fileid assigned by the Sender to identify this file's current registration. This is a unique number, systemwide, that identifies the instance of a file. That is, if a file is unregistered by name then registered again the Sender will assign it a new, ever increasing, file id
Unconfirmed/ Confirmed	Indicates whether or not the file has been completely confirmed by all addressed sites

Entry	Description
Confirmation State	The state of this file on the corresponding receiver. It can be one of: • *Not Confirmed* • *Delivered OK* – No install program run at the receiver • *No Client* – No client at the receiver • *Bad Client Directory* – cannot write into destination directory • *FInstall Error Code* – Install program returned an error • *FInstall Timed Out* – Install program never completed • *Error Changing Ownership* – Cannot set the owner of the delivered file to the same as the parent directory • *No Disk Space to Load* – Not enough space to initially load the file into the receiver's staging directory • *No Disk Space to Install* – Not enough space to move the file from its staging directory to the client's destination directory • *No Bandwidth* – The receiver's configuration does not permit receiving this file at its announced bit rate • *FAnnounce Rejected* – The FANNOUNCE application at the receiver rejected even loading this load • *Filtered Out* – The receiver has configured a filter mask which rejects loading this file • *Installed OK* – File delivered and install program completed successful
Errno	The error code returned by FInstall at the receiver. This is customer defined
Numloads	The number of load attempts, in whole or in part, needed to receive this file
Cnftime	The time, normalized to the Sender's time, this file was written into the client's destination directory at the receiver
Erasures	The number of missed packets during the first load attempt at the receiver

A sample package confirmation response that reflects a package addressed to 3 sites and confirmed by one is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE confirmationStatusResponse SYSTEM "http://ROCK-
SOFT05/dtd/pd/1.0/confirmationStatusResponse.dtd">
<confirmationStatusResponse>
<fileName value="4MB.txt"/>
<fileID value="30"/>
<unconfirmedStatus/>
<confirmationStatus>
  <site id="WINXP001"/>
  <cnfstate>Not Confirmed</cnfstate>
  <errno>0</errno>
  <numloads>0</numloads>
  <cnftime>...Not Confirmed...</cnftime>
  <erasures>0</erasures>
</confirmationStatus>
<confirmationStatus>
  <site id="WINXP002"/>
  <cnfstate>Not Confirmed</cnfstate>
  <errno>0</errno>
  <numloads>0</numloads>
  <cnftime>...Not Confirmed...</cnftime>
  <erasures>0</erasures>
</confirmationStatus>
<confirmationStatus>
  <site id="WINXP003"/>
  <cnfstate>Delivered OK</cnfstate>
  <errno>0</errno>
  <numloads>1</numloads>
  <cnftime>2009/12/18 13:20:15</cnftime>
  <erasures>0</erasures>
</confirmationStatus>
</confirmationStatusResponse>
```

Commands and Responses :
Poll DMB Receiver Health

1. *Command* - There is a single command for this transaction, named POLL_RECEIVER_HEALTH. The Sender issues a multicast ping (i.e., mping) to the subset of requested remotes, waits for responses, and aggregates all responses into a local flat-file log of the polling results.

Note that only one poll sequence can be outstanding at a time. Any requests received while a poll is in progress will be spooled, on the Sender, for single-threaded execution.

2. *Parameters* - The following table describes the specific parameters within the *POLL_RECEIVER_HEALTH* transaction.

Entry	Description
SOURCE_CLIENT	The source client name for this request
REQUEST_ID	The globally unique id chosen by the client to associate with this poll request
TIMEOUT	The number of seconds to wait for a response from receivers. Default of 60
RETRIES	The number of times to retry an mping to unresponsive receivers. Default of 0
DESTINATION	The destination field identifies the Receivers for which the mping is intended. The destination field contains the complete list of SiteIds to poll. The list of entries can also include one of the following two options: • *BROADCST*- A single line indicating that the mping is for all receivers. • *@GROUPNAME* - The Sender expands the group name into a list of Receiver SiteIds at parse time (i.e., when the transaction is posted). These are the SiteIds which will be mping'd

3. *Command XML Format* - The pollReceiverHealth command DTD is pollReceiverHealthCmd.dtd. A sample command of a BROADCST mping is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PollReceiverHealthCmd SYSTEM "../dtd/pd/1.0/pollReceiverHealthCmd.dtd">
<PollReceiverHealthCmd>
        <sourceClient name="ADMIN"/>
        <requestId>test</requestId>
        <timeout>60</timeout>
        <retries>1</retries>
        <receiverDestination>
                <destination/>
        </receiverDestination>
</PollReceiverHealthCmd>
```

4. *Response* - The pollReceiverHealth response DTD is response.dtd and contains:

Entry	Description
STATUS	Indicates: • PASS • FAIL
ERROR	For a failed transaction, indicates: <errno>

Commands and Responses :
Retrieve DMB Receiver Health

1. *Command* - There is a single command for this transaction, named *RETRIEVE_RECEIVER_HEALTH*. The Sender returns the results of the last *POLL_RECEIVER_HEALTH* operation

2. *Parameters* - The following table describes the specific parameters within the *RETRIEVE_RECEIVER_HEALTH* transaction

Entry	Description
SOURCE_CLIENT	The source client name for this request
REQUEST_ID	The globally unique id chosen by the client for which status is requested

3. *Command XML Format* - The retrieveReceiverHealth command DTD is retrieveReceiverHealthCmd.dtd. A sample command is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE retrieveReceiverHealthCmd SYSTEM "../dtd/pd/1.0/
retrieveReceiverHealthCmd.dtd">
<retrieveReceiverHealthCmd>
        <sourceClient name="ADMIN"/>
        <requestId>test</requestId>
</retrieveReceiverHealthCmd>
```

4. *Response* - The retrieveReceiverHealth response DTD is retrieveReceiverHealthResponse.dtd and contains:

Entry	Description
STATUS	Indicates: • PASS • FAIL
ERROR	For a failed transaction, indicates: <errno>
RESPONSES	The responses field consists of a list of receiver responses for the previous polling sequence. The format for the responses field entries is one remote response file contents per line. Each line begins with a SiteId followed by the IP address of the siteid and an optional free format ascii text string up to 1K characters in length. This string is provided by the remote in its reponse to the Sender. Its contents is remote defined

A sample retrieve Receiver health response follows. It shows responses from two Receivers, each with multiple IP addresses (multi-homed platforms) and additional health status information.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE retrieveReceiverHealthResponse SYSTEM ".../dtd/
pd/1.0/retrieveReceiverHealthResponse.dtd">
<retrieveReceiverHealthResponse>
<pass/>
<responses>
<siteId name="WINXP001">
1.1.1.5,10.4.4.4,Free Disk Space:2367289289, OS: XP
</siteId>
<siteId name="WINXP003">
1.1.1.1,10.4.4.9,Free Disk Space: 367833939, OS: XP
</siteId>
</responses>
</retrieveReceiverHealthResponse>
```

Appendix A : Transaction Error Codes

Transaction Error Codes

Errno	Description
1	Invalid Group name
2	Not all sites belong to this client
3	Client <client name> is not valid
4	Client link file <file name> not found
5	Can not link file <link file name> to <file name>
6	Client file <file name> not found
7	Stop_time is in the past <time>
8	The bit rate is higher than available bandwidth of: <number> bps
9	The bit rate specified is larger than provider's maximum allowed bit rate
10	The bit rate specified is smaller than the provider's minimum allowed bit rate
11	The bit rate specified is smaller than provider's minimum allowed bit rate
12	Invalid start time (stop time is before start time)
13	Client/file <client name>/<file name> previously NOT set for CONFIRM
14	Cannot add element to publisher: <client name> community: <community name> in CAS
15	Cannot delete file id: <number> from CAS
16	Invalid DESTINATION field, SiteId: <number>
17	Invalid timeout requested
18	Invalid number of retries requested
19	Poll sequence still outstanding
20	Invalid Request Id
21	SOURCE_CLIENT not an Administrator
22	Invalid community name
23	Invalid IPMC
24	IPMC Address already in use
25	Group name too long
26	Error in expanding group definition
27	Source client not found
28	Source client name too long
29	Source client does not have privileges
30	Target client name too long
31	Link file name too long
32	Invalid Default Stop Time
33	Invalid Bit Rate
34	Invalid destination SITE ID name length
35	Source client name still undefined
36	Community name too long
37	Undefined community parameter
38	Invalid Community name in envelope
39	Can't connect to Conditional Access Server
40	Undefined Access Level for the Community
41	EMail address too long
42	Invalid YEAR/MONTH/DAY Format

Errno	Description
43	Invalid YEAR/MONTH/DAY Format - not a leap year
44	Invalid HH:MM:SS Format
45	Can't convert YYYY/MM/DD HH:MM:SS to calendar time
46	Invalid destination client name length
47	Invalid Reservation ID
48	Invalid HH:MM Format
49	REPETITION string too long
50	INVALID REPETITION string
51	SERVICE string too long
52	INVALID SERVICE string
53	Invalid ACTUAL DURATION
54	Invalid BANDWIDTH
55	Invalid Default FEC
56	Maximum number of envelopes exceeded
57	Can not obtain file size
58	<token> too long
59	Envelope incomplete - missing DESTINATION
60	Missing BEGIN in DESTINATION field
61	Invalid Envelope Format
62	Invalid Destination comm-server name length
63	Invalid Envelope Format
64	Not all sites belong to client <client name>
65	Sites in envelope do not subscribe to client <client name>
66	ACTUAL_DURATION greater than 24 hours!
67	Invalid BEGIN found
68	BwResParser: Invalid keyword <string>
69	Invalid HH:MM Format
70	Invalid EMail address
71	Invalid <token>
72	Undefined community parameter
73	Invalid Client name for this Provider
74	Undefined operation
75	Undefined parameter in Community envelope
76	Incomplete envelope, missed Destination
77	<string> is a undefined community type
78	Unknown parameter in Status Reporting
79	Invalid IPMC length
80	Missing BEGIN in IPMC_ADDRESSES field
81	Invalid IPMC Format
82	Address <string> is not IPMC
83	Access Control Error: <string> does not subscribe to client <client name>
84	Too many sites to add and/or delete from this community, adding: <number>, removing: <number>, max allowed: <number>
85	Conditional Access Error. Can't redefine <community name> community <client name> client

Errno	Description
86	Database Error: Sites could not be deleted from community <number>
87	Database Error: <site name> could not be added to <number>
88	Too many sites to append to this community, count: <number>, max allowed: <number>
89	Conditional Access Error. Can't redefine <community name> community <client name> client
90	Too many sites to delete from this community, count: <number>, max allowed: <number>
91	Delete failed. Site <site name> does not belong to <community name> community <client name> client
92	Community <community name>, Client <client name> has children. Can't delete
93	Community <community name>, Client <client name> is root community. Can't delete
94	Conditional Access Error. Can't delete <community name> community <client name> client
95	Too many sites to add to this new community, count: <number> max allowed: <number>
96	Conditional Access Error: Could not create community <community name> client <client name>
97	Not all sites belong to this Provider
98	Client is not allowed to create communities
99	Default priority can not be greater then Maximum priority
100	ClientMinBitRate can not be greater then ClientMaxBitRate
101	ClientDefBitRate has to be between ClientMinBitRate and ClientMaxBitRate
102	Client <client name> is not valid
103	STARTTIME = STOPTIME and Expiration = AUTO ... not allowed
104	STARTTIME = STOPTIME .... not allowed
105	requestable file cannot be periodic
106	periodic file must have repetition value
107	RETRANSMIT or CONFIRM delivery assurance cannot have repetition value
108	REQUEST is only interactivity permitted with explicit community
109	PULL and BEST_EFFORT is not a valid combination
110	REQUEST and BEST_EFFORT is not a valid combination
111	PRICE parameter may only be used with an explicit community
112	The CONFIRM delivery assurance cannot be used with BROADCST destination option
113	EMAIL address not defined for EMAIL type of logging
114	SUCCESS expiration type works only with CONFIRMable packages
115	Confirmation Deadline can not be used with non confirmable package
116	Too early Confirmation Deadline
117	Envelope Incomplete: Missing SOURCE_CLIENT
118	Envelope Incomplete: Missing FILE_NAME
119	Envelope Incomplete: Missing DEST_FILE_NAME
120	Envelope Incomplete: Missing DESTINATION
121	must specify CLIENT
122	must specify REPETITION
123	must specify bandwidth
124	enddate can not be earlier than startdate
125	first reservation timeslot should be later than <time>
126	first reservation timeslot should be earlier than the last reservation slot
127	attempting to use default streamname
128	attempting to use default email address

Errno	Description
129	must specify reservation id
130	must specify one of the parameters to modify
131	Unable to validate bandwidth reservation envelope:
132	Cannot modify element for pub: <client name> from comm: <community name> to comm: <community name> in CAS
133	Error in CAS transmission, rejected
134	Can't recover after error in CAS transmission, rejected
135	Source file name too long
136	Client * NOT assigned to gateway for stream *
137	Requested bandwidth should be greater than minbitrate (* kbps) configured for client*
138	Requested bandwidth should be less than maxbitrate (* kbps) configured for client*
139	Unable to find *
140	Requested bandwidth exceeds bandwidth limits on gateway
141	Requested bandwidth exceeds per-provider bandwidth limits on shared gateway
142	Reservation id: * NOT FOUND
143	Cannot modify a cancelled reservation
144	Cannot modify a previously modified reservation
145	Cannot modify an expired reservation
146	ADMIN_ACCOUNT_ERR
147	Invalid FEC Overhead
148	Destination file name too long
149	Can not read envelope file
150	Invalid keyword
151	More than one group defined
152	Invalid Priority Value
153	PreProcess file name too long
154	Invalid Interactivity Value
155	Invalid descriptive name
156	Unknown Expiration Type
157	Unknown Envelope Delivery Type
255	XML Validation Error

Appendix B : Figures

Video File Extension Format

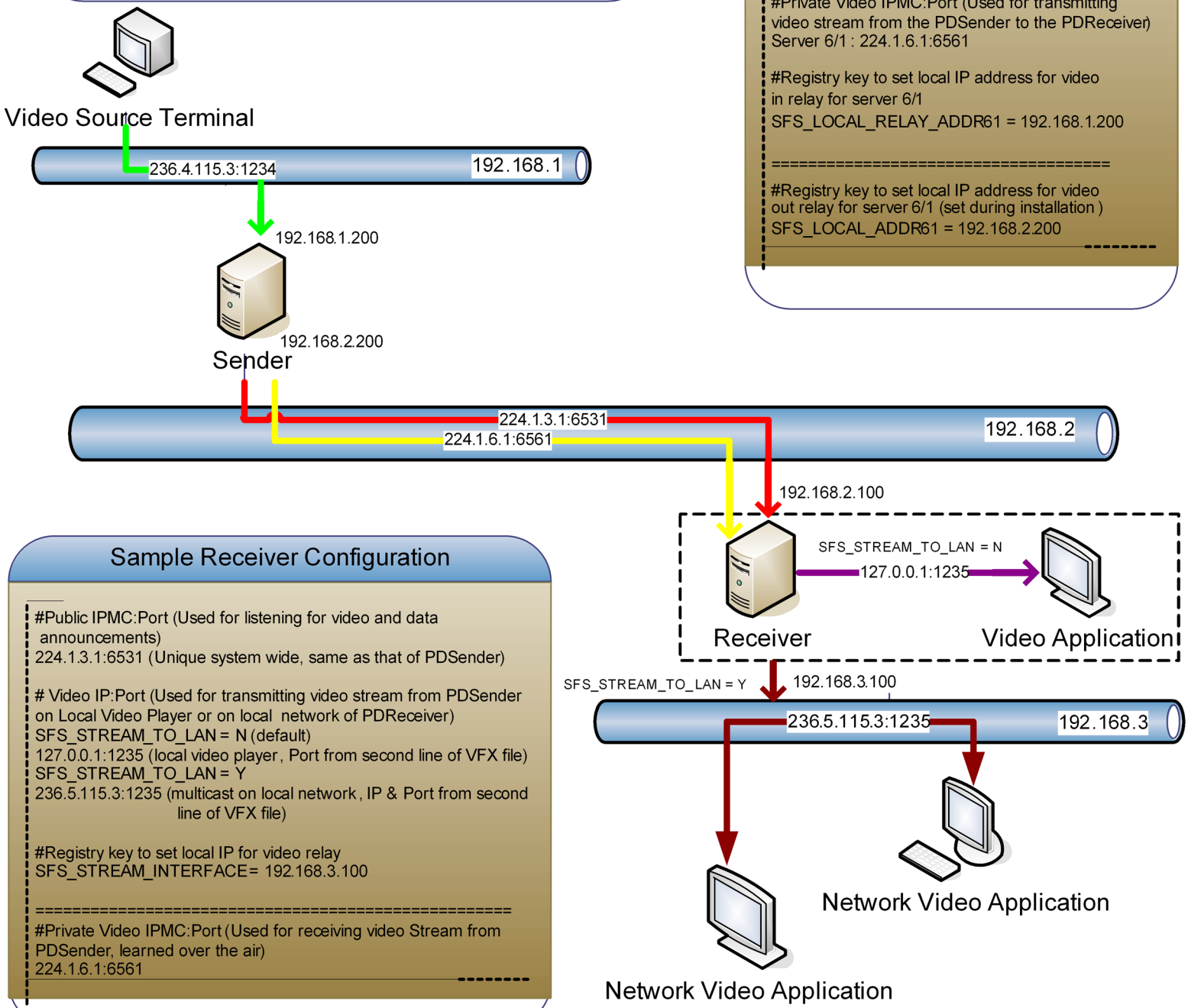
When registering files for transmission the Sender supports the "Video File Extension" format to designate the IPTV streaming parameters. A .vfx files designates the streaming video source to relay, the destination addressing and the number of seconds that the video should remain active once transmission begins. The following example network configuration includes the .vfx file format:

Sample VFX File registered at Sender

```
#Following is the IPMC:Port of the Streaming source (the Streamer)
236.4.115.3:1234
#Following is the destination IPMC:Port where the PDReceiver relays the stream
236.5.115.3:1235
#Following is the number of seconds to relay the stream to the receiver
240
```

Sample Sender Configuration

```
#Public IPMC:Port (Used for announcements)
Server 3/1 : 224.1.3.1:6531 (Unique IP & Port)

#Private Data IPMCs:Ports (Used for transmitting
 data files)
Server 3/2 : 224.1.3.2:6532 (Unique IP & Port)
Server 3/3 : 224.1.3.3:6533 (Unique IP & Port)

#Private Video IPMC:Port (Used for transmitting
video stream from the PDSender to the PDReceiver)
Server 6/1 : 224.1.6.1:6561

#Registry key to set local IP address for video
in relay for server 6/1
SFS_LOCAL_RELAY_ADDR61 = 192.168.1.200

==================================

#Registry key to set local IP address for video
out relay for server 6/1 (set during installation )
SFS_LOCAL_ADDR61 = 192.168.2.200
```



Sample Receiver Configuration

```
#Public IPMC:Port (Used for listening for video and data
 announcements)
224.1.3.1:6531 (Unique system wide, same as that of PDSender)

# Video IP:Port (Used for transmitting video stream from PDSender
on Local Video Player or on local  network of PDReceiver)
SFS_STREAM_TO_LAN = N (default)
127.0.0.1:1235 (local video player, Port from second line of VFX file)
SFS_STREAM_TO_LAN = Y
236.5.115.3:1235 (multicast on local network, IP & Port from second
                  line of VFX file)

#Registry key to set local IP for video relay
SFS_STREAM_INTERFACE= 192.168.3.100

================================================
#Private Video IPMC:Port (Used for receiving video Stream from
PDSender, learned over the air)
224.1.6.1:6561
```

Appendix C : Daily Logs

Daily Logs

The logs described in this appendix contain daily records of activity of various events throughout the life of user files. Each log name is formatted as <**LogFileName**><**MM**>-<**DD**> where MM is the month and DD is the day of the log. For example the parser log file for May 1, 2009 is named ParseLog05-01.

The package parser and scheduler logs are formatted with whitespace separated fields where each line contains a single log entry. The request and confirmation logs contain a fixed length record on each line with a single space separating the fields. The field lengths are provided below. All logs, if enabled by the Sender administrator, are stored by the Sender in the …/pkglog directory under a particular user's account. These logs can be accessed through the user's FTP conection.

The logs should be archived and purged on at least an annual basis to keep the number of log files from growing beyond a year's worth of logging information.

Daily Logs : Package Parser Log

This is an audit trail of each package registration for the user. A sample entry from *ParseLog05-01*, is:

```
00001 20090501115633 20090501121000 XYZCORP   0002805743 ....New  "2224555.mpg"
00001 20090501115636 20090501121000 XYZCORP   0002805744 ....New  "2224556.mpg"
00001 20090501115639 20090501121000 XYZCORP   0002805745 ....New  "2224557.mpg"
00001 20090501115642 20090501121000 XYZCORP   0002805746 ....New  "2224555.mpg"
```

Field	Meaning
00001	Parser log version
20090501115642	Time of registration
20090501121000	Requested package transmission time
XYZCORP	Owner of the package
0002805746	Unique package id
...New	Action. Values are: New, Update.
2224555.mpg	Package name (enclosed in quotes)

Daily Logs : Package Scheduler Log

This is an audit trail of all package transmission activity for the user. It shows all transmission starts, stops, and preemptions. A sample, from the file named *SchedLog05-01*, is:

```
00001 20090501120641 3/3 Transmit   XYZCORP   0002805619 "2196006.mpg"
00001 20090501120710 3/3 Finished   XYZCORP   0002805619 "2196006.mpg"
00001 20090501120710 3/3 Transmit   XYZCORP   0002805618 "2196218.mpg"
00001 20090501120824 3/3 Finished   XYZCORP   0002805618 "2196218.mpg"
```

Field	Meaning
00001	Scheduler log version
20090501120710	Timestamp of the action
3/3	Sender Transmit Stream
Finished	Action. Values are: Transmit, Preempt, Ack, Resume, Finished, Abort
XYZCORP	Owner of the package
0002805619	Unique package id
2196006.mpg	Package name

Daily Logs : **Request Log**

This is a comprehensive audit trail of each package request received from a remote. A sample, from the file named *Request01-11.log*, is:

20090111101742 “REQUEST “ 0000000003 TESTSITE
10.4.4.4 “Client File “ 224.1.3.1:6531 10.4.4.4:6531 ffffffffff
ffffffffffffffffffffffffffffffffffffffff3ffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffc00000000000000003fffffffffffffffffffffffffffff fffffffffffffff
fffffffffffffffffffffffffffffffffffff3fffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffc00000000000000003fffffffffffffffffffffffffffff

Field	Meaning	Field Length
20090111101742	Date and Time Request	14
“REQUEST”	“REQUEST” or “DUPLICATE REQUEST” or “ERROR”. A duplicate means that the file is already queued for transmission. An error means that the file is no longer registered in the Sender’s local database.	19
0000000003	Unique package id	10
TESTSITE	Site ID of the requesting remote	8
10.4.4.4	IP of the requesting Receiver	17
“Client File”	Client File or FEC. This is the type of file being requested. FEC requests the parity file needed to apply forward error correction	17
224.1.3.1:6531	The IPMC:Port or IP:Port the Receiver is listening to for the Broadcast Announcement for this package.	25

Field	Meaning	Field Length
224.1.3.1:6531	IP:Port which designates the interface the Receiver is listening to for the Broadcast Accouncement for this package. An IP Address of 0.0.0.0 indicates that the announcement can arrive on any number will always the same as the port where the receiver is listening to for the announcment. This field is relevant on a multi-homed Receiver which may be configured to accept initial transision of a package through one local adapter and the retransmission through another adapter. Any “separate retransmit channel” configuration like this must match on the Sender and every Receiver in the network.	25
fffffffffffffffffffffffffff ffffffffffffffffffff3fffff fffffffffffffffffffffffffff fffffffffffffffffffffffffff fffffffffffffffffffffffffff fffffffffffffffffffffffffff fffffffffffffffffffffffffff fffffffffffffffffffc0000 0000000000003fffff ffffffffff	The 1k bit request mask. This indicates the segments of the package needed by this remote. The mask should be read from right to left (i.e., the rightmost bit corresponds to the beginning of the file) with a 9 bit indicating a needed segment. If the “REQUEST” field is ERROR then this field is set to UNKNOWN.	256
fffffffffffffffffffffffffff ffffffffffffffffffff3fffff fffffffffffffffffffffffffff fffffffffffffffffffffffffff fffffffffffffffffffffffffff fffffffffffffffffffffffffff fffffffffffffffffffffffffff fffffffffffffffffffc0000 0000000000003fffff ffffffffff	The current 1k bit transmission mask. This indicates the current set of segments queued for transmission after application of this request mask. It reflects the cumulative set of masks from each requesting remote up until the file is transmitted. If the “REQUEST” field is ERROR then this field is set to UNKNOWN.	256

This is a comprehensive audit trail of each package confirmation received from a remote. A sample, from the file named *Confirm01-28.log*, is:

```
20090128000005 0000006132   REDHAT02        10.4.4.30
"Delivered OK          " 2009/01/28 00:00:05 0000000001Loads
0000000000Gaps 0000000000Errno
```

Field	Meaning	Field Length
20090128000005	Date and Time of Request	14
0000006132	Unique package id	10
REDHAT02	Site ID of the requesting Receiver	8
10.4.4.30	IP of the confirming Receiver	17
"Delivered OK"	One of: *Delivered OK*, *No Client*, *Bad Client Directory*, *FInstall Error Code*, *FInstall Timed Out*, *Error Changing Ownership*, *Preprocessor Error*, *No Disk Space to Install*, *No Bandwidth*, *FAnnounce Rejected*, *Filtered Out*, or *Installed OK*	26
2009/01/28 0:00:05	The delivered time at the Receiver. This is the time (normalized to match the Sender's time) that the file was written into the client's destination directory on the remote.	19
0000000001Loads	The number of full or partial load attempts at the remote	15
0000000000Gaps	The number of gaps in the incoming file during its first load attempt. This can be used to monitor the health of an individual Receiver or compare the reception performance across multiple Receivers.	14
0000000000Errno	The local errno as reported by an external application at the remote. For instance, FINSTALL can provide an error exit status when installing a delivered file. This status will be returned as the errno value in the file's confirmation.	15

Appendix D : Video and Audio Multicasting

Video and Audio Multicasting

The Sender supports multicast streaming of locally stored MPEG1, MPEG2 and MPEG4 files, including audio files, for real-time rendering on compatible downstream devices.

The following table describes supported file formats:

Media Container	Video Encoding	Audio Encoding	Protocol
MPEG Transport Stream	MPEG1, MPEG2, MPEG4.1 or MPEG4.10(h264)	MPEG Audio	UDP Multicast
Video Only	MPEG1 or MPEG2		RTP Multicast
Audio Only		MPEG Audio (including MP3)	RTP Multicast
MPEG Program Stream	MPEG1 or MPEG2	Mpeg Audio	RTP Multicast

The user instructs the Sender to stream the locally stored file (posted to the Sender through the user's FTP account) by submitting a specially formatted envelope file into the user's …*/envelope* directory. An example envelope file is:

```
BEGIN
SOURCE_CLIENT ADMIN
FILE_NAME VideoFile.mpg
DEST_FILE_NAME VideoFile.mpg
DESTINATION
 BEGIN
 END
END
#Then, create the new video registration to begin
now
BEGIN
SOURCE_CLIENT ADMIN
FILE_NAME VideoFile.mpg STREAM
DEST_FILE_NAME VideoFile.mpg
INTERACTIVITY PUSH
ASSURANCE BEST_EFFORT 1
PRIORITY HIGH
DESTINATION
 BEGIN
 BROADCST,DEFAULT
END
END
```

Note the presence of the tag STREAM after the file name. This tag tells the Sender to multicast stream the file to the network rather than transmitting it as a file delivery. The Sender administrator will configure the internal streaming server and publish the IPMC:Port used as the streaming destination. Multicast capable downstream clients (e.g., VLC on a PC, a Set-Top-Box) that can receive and play the video or audio stream can tune to the published IPMC:Port for reception.

The Sender automatically streams such files at the bit rate designated inside the file and, if necessary, scales back the rate of any non-stream transmissions (i.e., regular file deliveries) in progress to prevent exceeding any configured bandwidth constraints (e.g., time-of-day rates often used to limit the amount of bandwidth a Sender is authorized to consume at particular times during the week).