Hewlett Packard
Enterprise

HPE Media Server

Software Version: 11.1

Media Server Administration Guide

Document Release Date: June 2016
Software Release Date: June 2016

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise Development LP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HPE required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

HPE Big Data Support provides prompt and accurate support to help you quickly and effectively resolve any issue you may encounter while using HPE Big Data products. Support services include access to the Customer Support Site (CSS) for online answers, expertise-based service by HPE Big Data support engineers, and software maintenance to ensure you have the most up-to-date technology.

To access the Customer Support Site

- go to https://customers.autonomy.com

The Customer Support Site includes:

- **Knowledge Base**. An extensive library of end user documentation, FAQs, and technical articles that is easy to navigate and search.
- **Support Cases**. A central location to create, monitor, and manage all your cases that are open with technical support.
- **Downloads**. A location to download or request products and product updates.
- **Requests**. A place to request products to download or product licenses.

To contact HPE Big Data Customer Support by email or phone

- go to http://www.autonomy.com/work/services/customer-support

Support

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, visit the Knowledge Base on the HPE Big Data Customer Support Site. To do so, go to https://customers.autonomy.com, and then click **Knowledge Base**.

The Knowledge Base contains documents in PDF and HTML format as well as collections of related documents in ZIP packages. You can view PDF and HTML documents online or download ZIP packages and open PDF documents to your computer.

About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Contents

Part I: Getting Started

This section describes how to set up, configure, and run Media Server.

Chapter 1: Introduction

This section provides an overview of Media Server.

Media Server

Images and video are examples of unstructured information that represent a vast quantity of data. Media Server enables you to maximize the utility of this data by extracting meaningful information about its content.

For example, Media Server can:

- run optical character recognition, to read text in a scanned document or subtitles in a video.
- identify a person who appears in an image or video by matching their face to a database of known faces.
- identify logos and other objects when they appear in images and video.
- identify the exact time of a scene change in video.
- determine whether a video contains speech, and convert the speech into text.

You can deploy Media Server for broadcast monitoring purposes, to identify when individuals or organizations appear in television broadcasts and extract information about these appearances. You might also use Media Server to catalog an existing archive of audio and video clips.

In security and surveillance deployments, Media Server can help security personnel. Human operators can become overwhelmed by the amount of data available from CCTV cameras. Media Server reduces the operator's workload with automatic processing, such as reading the number plates on vehicles and automatically identifying suspicious events.

Media Server can transform the metadata that it extracts into many output formats, and send the data to many systems.

Media Server can be used to analyze images, video, and audio for a custom application that you develop. You can also use Media Server as part of a larger IDOL deployment, to run analysis on files that your connectors extract from your organization's data repositories, from web sites, and from social media. If you index information from Media Server into IDOL Server, you can use IDOL to search within your media. Searches will return images such as scanned documents that contain the search terms. You can search ingested video for specific events, such as the appearance of a particular speaker or discussion of a particular subject. If you are running Media Server for broadcast monitoring you can run sentiment analysis to determine whether the appearance of an individual or organization contained positive sentiment. If you are cataloging clips, you can use IDOL to categorize the clips into a custom taxonomy.

The following sections provide more information about Media Server features.

Ingest Media

Ingestion is the process of bringing media into Media Server so that it can be processed and analyzed. Media Server can ingest the following media:

- image files.
- office documents such as PDF files that contain embedded images.
- video files.
- video from IP streams. Many devices, for example IP cameras, network encoders, and IPTV devices can produce IP streams.
- video from cameras and third-party video management systems.

Analyze Media

Media Server can run many types of analysis, including:

- automatic number plate recognition
- barcode recognition
- color analysis
- face detection, face recognition, demographic analysis and expression analysis
- scene analysis
- keyframe extraction
- object recognition
- image classification
- optical character recognition
- speaker identification
- speech-to-text

For more information about the types of analysis that you can run, see "Analyze Media" on page 112.

Encode and Stream Video

If you ingest video, Media Server can write the video to disk. You can also configure rolling buffers, fixed-size storage areas on disk where the oldest content is discarded to make space for the latest. For example, if you deploy Media Server for video surveillance, you could configure a rolling buffer to store the last seven days of video from a camera.

You can also create a live UDP stream of the content ingested by Media Server.

Event Stream Processing

You can configure Media Server to filter, deduplicate, and find combinations of events in analysis results. For example, you could use Event Stream Processing (ESP) rules to identify events where the text "Breaking News" appears in a television broadcast *and* the newsreader speaks the words "election

results" within 10 seconds. You can add custom logic by writing scripts using the Lua scripting language.

Output Information

Media Server can output the metadata that it extracts to many formats and systems, including:

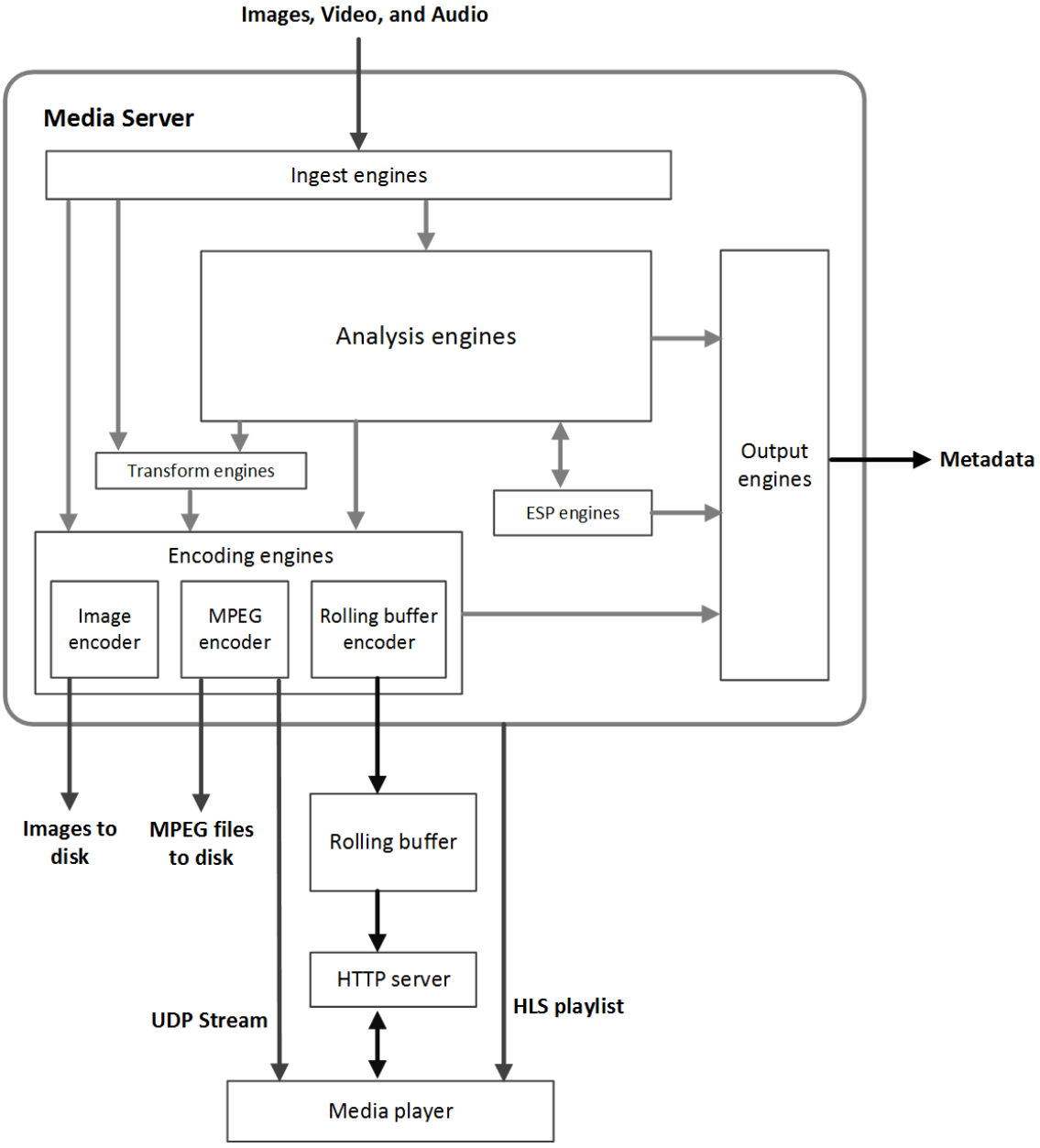
- Connector Framework Server (CFS)
- IDOL Server
- Vertica databases
- XML
- Milestone XProtect

OEM Certification

Media Server works in OEM licensed environments.

Media Server Architecture

The following diagram shows the architecture of Media Server.



A Media Server *engine* performs a single function such as ingestion or analysis. There are several types of engine:

- **Ingest engines** bring information into Media Server for processing. Images might be decoded or extracted from documents; video must be demuxed and decoded into audio, video, and metadata.

- **Analysis engines** run analysis on ingested media to extract information about its content. Each engine performs a different type of analysis.
- **Event stream processing engines** can be used to introduce additional custom logic into analysis. For example, you can filter or deduplicate the information produced during analysis. Event stream processing can change the schema of the data.
- **Transform engines** transform data, but do not modify its schema. For example, you can change the size of video frames before sending them to the image encoder.
- **Encoding engines** write a copy of ingested video to disk, or encode it into different formats. Some encoding engines, such as the MPEG encoder, can produce a live UDP stream that you can view in a media player.
- **Output engines** convert the metadata produced by Media Server into different formats and send it to other systems such as IDOL Server or a Vertica database.

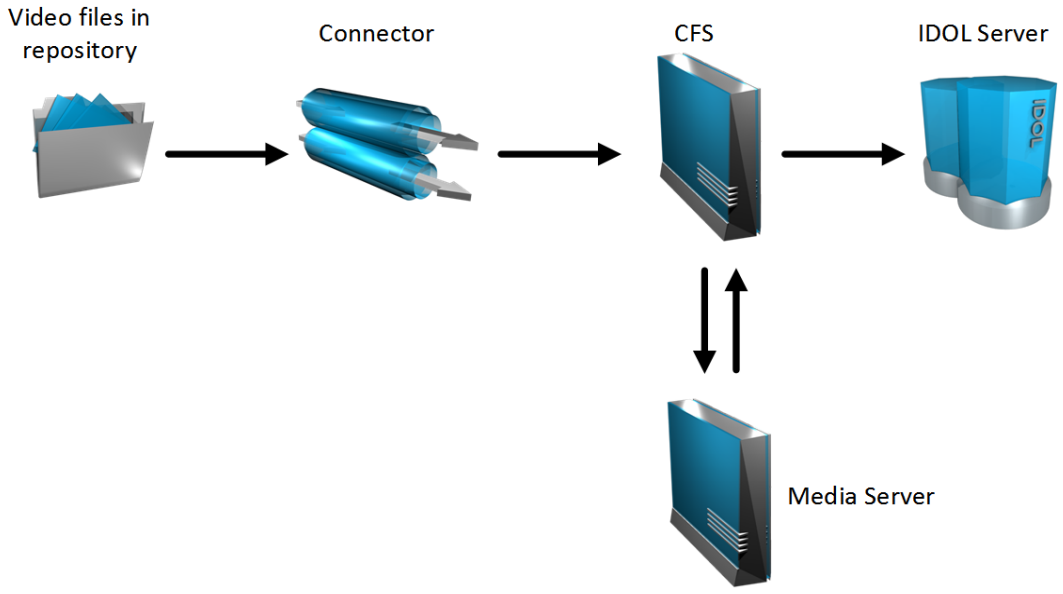
Information is passed between engines in the server. For example, an ingest engine decodes video and produces video frames and uncompressed audio for the other engines to use. Analysis engines run analysis on the decoded data. Output engines take information from the analysis engines and index it into other systems, such as IDOL Server.

HPE IDOL

Media Server is one of the components in HPE's *Intelligent Data Operating Layer* (IDOL). You can use Media Server independently or as part of a larger IDOL system.

You can use Media Server independently by writing a custom application that communicates with Media Server. Media Server accepts commands over HTTP and returns responses in XML format. You can also use the Autonomy Content Infrastructure (ACI) Client API to develop a custom application.

In a typical IDOL deployment, IDOL Connectors retrieve information from your data repositories for indexing into IDOL Server. You can configure your Connector Framework Server (CFS) to send images and video to Media Server and request one or more analysis operations. Media Server returns the results of the analysis operations to CFS, which enriches the information indexed into IDOL Server.



For example, a repository might contain video clips that you want to search or categorize. You could configure CFS to send the video to Media Server and request analysis such as face detection, face recognition, object recognition, keyframe extraction and optical character recognition. Media Server returns information about the video content to CFS, which might perform additional operations, such as Eduction, before indexing the information into IDOL Server.

For more information about IDOL, refer to the *IDOL Getting Started Guide*.

Related Documentation

The following documents provide more details on Media Server.

- *Media Server Reference*

  The *Media Server Reference* describes the ACI actions and configuration parameters that you can use with Media Server. For information about how to view the reference, see "Display Online Help" on page 65.
- *Media Server Scene Analysis Training Utility User Guide*

  The *Media Server Scene Analysis Training Utility User Guide* describes how to train Media Server to perform Scene Analysis.
- *Speech Server Administration Guide*

  Some of the analysis operations available through Media Server, for example speech-to-text and speaker identification, require an IDOL Speech Server. The *Speech Server Administration Guide* describes how to install, configure, train, and optimize your Speech Server.
- *Media Management and Analysis Platform Installation Guide*

  The HPE Media Management and Analysis Platform (MMAP) is a media analytics platform designed for viewing and searching video footage coming from a variety of sources, typically CCTV surveillance camera footage and broadcast footage from IP streams. The *Media Management and Analysis Platform Installation Guide* provides more information about MMAP.
- *License Server Administration Guide*

  This guide describes how to use a License Server to license multiple services.

Chapter 2: Install Media Server

This section describes how to install Media Server.

System Requirements

The system requirements for Media Server depend on the data that is being ingested and the processing tasks that are configured. HPE recommends the following minimum hardware specifications:

- A minimum of two dedicated CPUs
- 4 GB RAM

If you deploy multiple Media Servers across multiple machines, ensure that all of the machines synchronize their clocks with a time server.

Supported Platforms

- Windows x86 64
- Linux x86 64

The most fully tested versions of these platforms are:

Windows

- Windows Server 2012
- Windows Server 2008
- Windows 7

Linux

- Ubuntu 14.04
- Ubuntu 12.04
- CentOS 6

Install Media Server on Windows

Use the following procedure to install Media Server on Microsoft Windows operating systems, by using the IDOL Server installer.

The IDOL Server installer provides the major IDOL components. It also includes License Server, which Media Server requires to run.

To install Media Server

1. Double-click the appropriate installer package:

   `IDOLServer_`*`VersionNumber`*`_`*`Platform`*`.exe`

   where:

`VersionNumber`	is the product version.
`Platform`	is your software platform.

   The Setup dialog box opens.
2. Click **Next**.

   The License Agreement dialog box opens.
3. Read the license agreement. Select **I accept the agreement**, and then click **Next**.

   The Installation Directory dialog box opens.
4. Specify the directory to install Media Server (and optionally other components such as License Server) in. By default, the system installs on `C:\HewlettPackardEnterprise\IDOLServer-`*`VersionNumber`*. Click  to choose another location. Click **Next**.

   The Installation Mode dialog box opens.
5. Select **Custom**, and then click **Next**.

   The License Server dialog box opens. Choose whether you have an existing License Server.

   - To use an existing License Server, click **Yes**, and then click **Next**. Specify the host and ACI port of your License Server, and then click **Next**.
   - To install a new instance of License Server, click **No**, and then click **Next**. Specify the ports that you want License Server to listen on, and then type the path or click  and navigate to the location of your HPE license key file (`licensekey.dat`), which you obtained when you purchased Media Server. Click **Next**.

   The Component Selection dialog box opens.
6. Click **Next**.

7. Select the check boxes for the components that you want to install, and specify the port information for each component, or leave the fields blank to accept the default port settings.

   For Media Server you must specify the following information:

ACI Port	The port that you want Media Server to listen on, for ACI actions.
Service Port	The port that you want Media Server to listen on, for service actions.

   Click **Next** or **Back** to move between components.
8. After you have specified your settings, the Summary dialog box opens. Verify the settings you made and click **Next**.

   The Ready to Install dialog box opens.
9. Click **Next**.

   The Installing dialog box opens, indicating the progress of the installation. If you want to end the installation process, click **Cancel**.
10. After installation is complete, click **Finish** to close the installation wizard.

Install an IDOL Component as a Service on Windows

On Microsoft Windows operating systems, you can install any IDOL component as a Windows service. Installing a component as a Windows service makes it easy to start and stop the component, and you can configure a component to start automatically when you start Windows.

Use the following procedure to install Media Server as a Windows service from a command line.

To install a component as a Windows service

1. Open a command prompt with administrative privileges (right-click the icon and select **Run as administrator**).
2. Navigate to the directory that contains the component that you want to install as a service.
3. Send the following command:

   ```
   Component.exe -install
   ```

   where `Component.exe` is the executable file of the component that you want to install as a service.

   The `-install` command has the following optional arguments:

`-start {[auto] | [manual] | [disable]}`	The startup mode for the component. **`Auto`** means that Windows services automatically starts the component. **`Manual`** means that you must start the service manually. **`Disable`** means that you cannot start the service. The default option is **`Auto`**.
`-username UserName`	The user name that the service runs under. By default, it uses a local system account.
`-password Password`	The password for the service user.
`-servicename ServiceName`	The name to use for the service. If your service name

	contains spaces, use quotation marks (") around the name. By default, it uses the executable name.
`-displayname DisplayName`	The name to display for the service in the Windows services manager. If your display name contains spaces, use quotation marks (") around the name. By default, it uses the service name.
`-depend Dependency1 [,Dependency2 ...]`	A comma-separated list of the names of Windows services that Windows must start before the new service. For example, if you are installing a Community component, you might want to add the Agentstore component and Content component as dependencies.

For example:

```
content.exe -install -servicename ContentComponent -displayname "IDOL Server
Content Component" -depend LicenseServer
```

After you have installed the service, you can start and stop the service from the Windows Services manager.

When you no longer require a service, you can uninstall it again.

To uninstall an IDOL Windows Service

1. Open a command prompt.
2. Navigate to the directory that contains the component service that you want to uninstall.
3. Send the following command:

   ```
   Component.exe -uninstall
   ```

   where `Component.exe` is the executable file of the component service that you want to uninstall.

   If you did not use the default service name when you installed the component, you must also add the `-servicename` argument. For example:

   ```
   Component.exe -uninstall -servicename ServiceName
   ```

Install Media Server on UNIX

Use the following procedure to install Media Server in text mode on UNIX platforms.

To install Media Server on UNIX

1. Open a terminal in the directory in which you have placed the installer, and enter the following command:

   ```
   ./IDOLServer_VersionNumber_Platform.exe --mode text
   ```

   where:

VersionNumber	is the product version
Platform	is the name of your UNIX platform

Note: Ensure that you have execute permission for the installer file.

The console installer starts and displays the Welcome screen.

2. Read the information and then press the `Enter` key.

   The license information is displayed.

3. Read the license information, pressing `Enter` to continue through the text. After you finish reading the text, type **`Y`** to accept the license terms.
4. Type the path to the location where you want to install the servers, or press `Enter` to accept the default path.

   The Installation Mode screen is displayed.

5. Press `2` to select the Custom installation mode.

   The License Server screen opens. Choose whether you have an existing License Server.

   - To use an existing License Server, type **`Y`**. Specify the host and port details for your License Server (or press `Enter` to accept the defaults), and then press `Enter`. Go to Step 7.
   - To install a new instance of License Server, type **`N`**.

6. If you want to install a new License Server, provide information for the ports that the License Server uses.

   a. Type the value for the ACI Port and press `Enter` (or press `Enter` to accept the default value).

ACI Port	The port that client machines use to send ACI actions to the License Server.

   b. Type the value for the Service Port and press `Enter` (or press `Enter` to accept the default value).

Service Port	The port by which you send service actions to the License Server. This port must not be used by any other service.

   c. Type the location of your HPE license key file (`licensekey.dat`), which you obtained when you purchased Media Server. Press `Enter`.

7. The Component Selection screen is displayed. Press `Enter`. When prompted, type **`Y`** for the components that you want to install. Specify the port information for each component, and then press `Enter`. Alternatively, leave the fields blank and press `Enter` to accept the default port settings.

   For Media Server you must specify the following information:

ACI Port	The port that you want Media Server to listen on, for ACI actions.
Service Port	The port that you want Media Server to listen on, for service actions.

   Note: These ports must not be used by any other service.

The Init Scripts screen is displayed.

8. Type the user that the server should run as, and then press `Enter`.

   Note: The installer does not create this user. It must exist already.

9. Type the group that the server should run under, and then press `Enter`.

   Note: If you do not want to generate init scripts for installed components, you can simply press `Enter` to move to the next stage of the installation process without specifying a user or group.

   The Summary screen is displayed.

10. Verify the settings that you made, then press `Enter` to begin installation.

    The Installing screen is displayed.

    This screen indicates the progress of the installation process.

    The Installation Complete screen is displayed.

11. Press `Enter` to finish the installation.

Install an IDOL Component as a Service on Linux

On Linux operating systems, you can configure a component as a service to allow you to easily start and stop it. You can also configure the service to run when the machine boots. The following procedures describe how to install Media Server as a service on Linux.

Note: To use these procedures, you must have `root` permissions.

Note: When you install Media Server on Linux, the installer prompts you to supply a user name to use to run the server. The installer populates the init scripts, but it does not create the user in your system (the user must already exist).

The procedure that you must use depends on the operating system and boot system type.

- For Linux operating system versions that use systemd (including CentOS 7, and Ubuntu version 15.04 and later), see "Install a Component as a Service for a systemd Boot System" below.
- For Linux operating system versions that use System V, see "Install a Component as a Service for a System V Boot System" on the next page.

Install a Component as a Service for a systemd Boot System

To install an IDOL component as a service

1. Run the appropriate command for your Linux operating system environment to copy the init scripts to your `init.d` directory.

- Red Hat Enterprise Linux (and CentOS)

  ```
  cp IDOLInstallDir/scripts/init/systemd/componentname
  /etc/systemd/system/componentname.service
  ```

- Debian (including Ubuntu):

  ```
  cp IDOLInstallDir/scripts/init/systemd/componentname
  /lib/systemd/system/componentname.service
  ```

componentname is the name of the init script that you want to use, which is the name of the component executable (without the file extension).

For other Linux environments, refer to the operating system documentation.

2. Run the following commands to set to appropriate access, owner, and group permissions for the component:

   - Red Hat Enterprise Linux (and CentOS)

     ```
     chmod 755 /etc/systemd/system/componentname
     chown root /etc/systemd/system/componentname
     chgrp root /etc/systemd/system/componentname
     ```

   - Debian (including Ubuntu):

     ```
     chmod 755 /lib/systemd/system/componentname
     chown root /lib/systemd/system/componentname
     chgrp root /lib/systemd/system/componentname
     ```

   For other Linux environments, refer to the operating system documentation.

   where *componentname* is the name of the component executable that you want to run (without the file extension).

3. (Optional) If you want to start the component when the machine boots, run the following command:

   ```
   systemctl enable componentname
   ```

Install a Component as a Service for a System V Boot System

To install an IDOL component as a service

1. Run the following command to copy the init scripts to your `init.d` directory.

   ```
   cp IDOLInstallDir/scripts/init/systemv/componentname /etc/init.d/
   ```

   where *componentname* is the name of the init script that you want to use, which is the name of the component executable (without the file extension).

2. Run the following commands to set to appropriate access, owner, and group permissions for the component:

```
chmod 755 /etc/init.d/componentname
chown root /etc/init.d/componentname
chgrp root /etc/init.d/componentname
```

3. (Optional) If you want to start the component when the machine boots, run the appropriate command for your Linux operating system environment:
   - Red Hat Enterprise Linux (and CentOS):

     ```
     chkconfig --add componentname
     chkconfig componentname on
     ```

   - Debian (including Ubuntu):

     ```
     update-rc.d componentname defaults
     ```

   For other Linux environments, refer to the operating system documentation.

Install Media Server from the ZIP Package

HPE recommends that you install Media Server using the IDOL Server installer (see "Install Media Server on Windows" on page 22 or "Install Media Server on UNIX" on page 24). However, if necessary, you can install Media Server from the ZIP package.

To install Media Server from the ZIP package (on Windows)

1. Extract the contents of the ZIP package to your chosen installation directory.
2. Install all of the redistributables located in the `runtime` folder. These install prerequisites that are necessary to run Media Server.
3. Modify the parameters in the `[License]` section of the Media Server configuration file so that Media Server can query your License Server. For information about how to do this, see "Configure the License Server Host and Port" on page 49.
4. You can now start Media Server. You can install Media Server as a Windows service and then start and stop the service. For information about how to do this, see "Install an IDOL Component as a Service on Windows" on page 23.

To install Media Server from the ZIP package (on Linux)

1. Extract the contents of the ZIP package to your chosen installation directory.
2. Modify the parameters in the `[License]` section of the Media Server configuration file so that Media Server can query your License Server. For information about how to do this, see "Configure the License Server Host and Port" on page 49.
3. You can now start Media Server. To start Media Server, run the start script `start.sh`, and to stop Media Server run the stop script, `stop.sh`. These scripts are included in the Media Server installation directory.

Set up a Database to Store Training Data

Media Server uses a database to store information that it requires for recognition operations, such as face recognition, object recognition, or image classification. Media Server can be configured to use an internal database file or an external database hosted on a database server.

The default configuration supplied with Media Server uses an internal database and using this type of database requires no additional configuration.

HPE recommends that you use a database hosted on an external database server for the following reasons:

- **Better performance**. A database hosted on an external database server is likely to achieve significantly higher performance when your Media Server is used by multiple users and many training actions are sent to the Media Server simultaneously.
- **Sharing training data**. If you analyze large numbers of images and videos you can spread the load across multiple Media Servers. Multiple Media Servers can share a database hosted on an external database server so that all of the Media Servers use the same training data and you only need to maintain one database. Sharing an internal database is not supported.
- **Improved handling of concurrent requests**. When Media Server modifies data in an internal database file, it can lock the file. Any requests that need to modify the database at the same time might fail, especially if the file is locked for a significant amount of time (such as when you add large amounts of training). An external database hosted on a database server is able to handle concurrent requests.

Use the Internal Database

The default configuration supplied with Media Server stores training data in a database file (`mediaserver.db`, in the installation directory). If this file does not exist, Media Server creates it when you use an action that requires a database.

You can move or rename the database file but you will then need to change your configuration file to match the new file path.

To use an internal database

1. Open the Media Server configuration file in a text editor.
2. In the `[Database]` section, check that the `DatabaseType` configuration parameter is set to **`internal`**. This is the default setting and specifies that Media Server uses an internal database.
3. In the `[Paths]` section, set the `DatabasePath` parameter to the path and file name of the database file. If this file does not exist, Media Server creates an empty database at this location.
4. Save and close the configuration file.
5. Restart Media Server for your changes to take effect.

Use an External Database

This section describes how to set up an external database and configure Media Server to use that database.

Supported External Databases

The following table lists supported platforms and databases that you can use to store training data:

Media Server Operating System	Supported Databases
Windows (64-bit)	• PostgreSQL version 9.1.x with PostgreSQL ODBC driver 09.03.0300 or later. • MySQL versions 5.x with MySQL ODBC Connector 5.3.x.
CentOS 6	• PostgreSQL version 9.1.x with PostgreSQL ODBC driver 09.03.0300 or later and unixODBC version 2.2.14 or later. • MySQL versions 5.x with MySQL ODBC Connector 5.1.x and unixODBC version 2.2.14 or later.

Note: Other platforms might work but have not been tested.

Tip: You can also store asynchronous action queues in a database. If you want to use the same database server to store training data and action queues, review the database requirements in the section "Store Action Queues in an External Database" on page 74, because the requirements for storing action queues might be different.

Set Up a PostgreSQL Database on Windows

To use Media Server with a PostgreSQL database, you must download and install a PostgreSQL server and ODBC driver, and configure Media Server to connect to the database through the driver.

The procedure describes setting up the database server using the psql command-line tool. If you prefer, you can use the pgAdmin graphical user interface. For more information, refer to the pgAdmin documentation on www.pgadmin.org.

To set up a PostgreSQL Media Server database on Windows

1. Download and install a PostgreSQL server. For instructions, refer to the PostgreSQL documentation on www.postgresql.org.
   - Ensure that the installation includes the PostgreSQL Unicode ODBC driver.
   - During installation, set up a user account with superuser privileges.

   Note: Once installed, the PostgreSQL server appears in the Services tab in Windows Task

Manager.

2. Add the PostgreSQL bin directory path to the `PATH` environmental variable.

   Note: This step enables you to use the command `psql` to start the PostgreSQL command-line tool (psql) from the Windows Command Prompt. If the directory path is not added to the `PATH` variable, you must specify the psql.exe file path in the Command Prompt to start psql.

3. Open the psql command-line tool:
   a. In the Windows Command Prompt, run the command:

      ```
      psql -U userName
      ```

   b. Enter your password when prompted.
4. Run a `CREATE DATABASE` command to create a new database. Specify the following database settings.

Database name	Any name.
Encoding	Must be Unicode–either UTF8 or UCS2.
Collation	Any that is compatible with the encoding.
Locale	Any that is compatible with the encoding.

   For example:

   ```
   CREATE DATABASE myDatabase WITH ENCODING 'UTF8' LC_COLLATE='English_United Kingdom' LC_CTYPE='English_United Kingdom';
   ```

5. Connect to the new database using the command:

   ```
   \c databaseName
   ```

6. Run the `postgres.sql` script provided in the Media Server installation directory. This script sets up the database schema that Media Server requires. The schema is inserted inside the `public` schema.
   a. HPE recommends running the following command to ensure that the script stops running if it encounters an error:

      ```
      \set ON_ERROR_STOP on
      ```

   b. Run the script using the command:

      ```
      \i 'path/postgres.sql'
      ```

      where *path* is the script file path.

      Note: Replace backslashes in the file path with forward slashes. The psql command-line tool does not recognize backslashes in file paths.

7. Grant privileges to the user that Media Server will connect as. Required privileges are:

Database	Create Temporary Tables

All tables	Select, Insert, Update, Delete
All functions and stored procedures	Execute

If security is not a consideration, grant all privileges.

For example:

```
GRANT TEMP ON DATABASE databaseName TO userName;
```

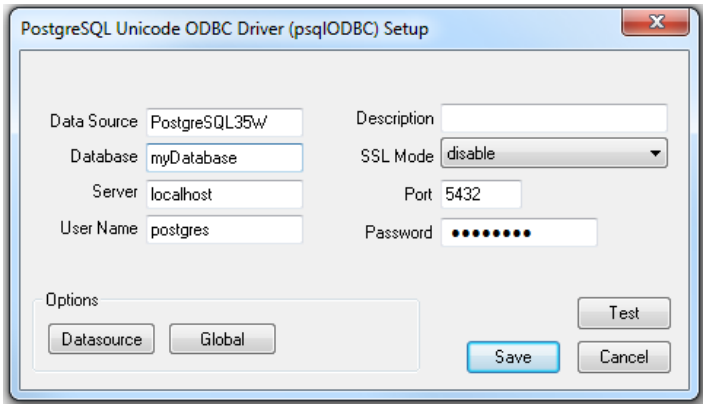
```
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO
userName;
```

```
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO userName;
```

where,

databaseName	is the name of the database that you created.
userName	is the user name that Media Server will connect as.

8. Open the Data Sources (ODBC) program:
   a. In the Windows Control Panel, click **System and Security**.
      The System and Security window opens.
   b. Click **Administrative Tools**.
      The Administrative Tools window opens.
   c. Double-click **Data Sources (ODBC)**.
      The ODBC Data Source Administrator dialog box opens.
9. In the User DSN tab, click **Add...** .
   The Create New Data Source dialog box opens.
10. Select the PostgreSQL Unicode driver from the list and click **Finish**.
    The PostgreSQL Unicode ODBC Driver (psqlODBC) Setup dialog box opens.



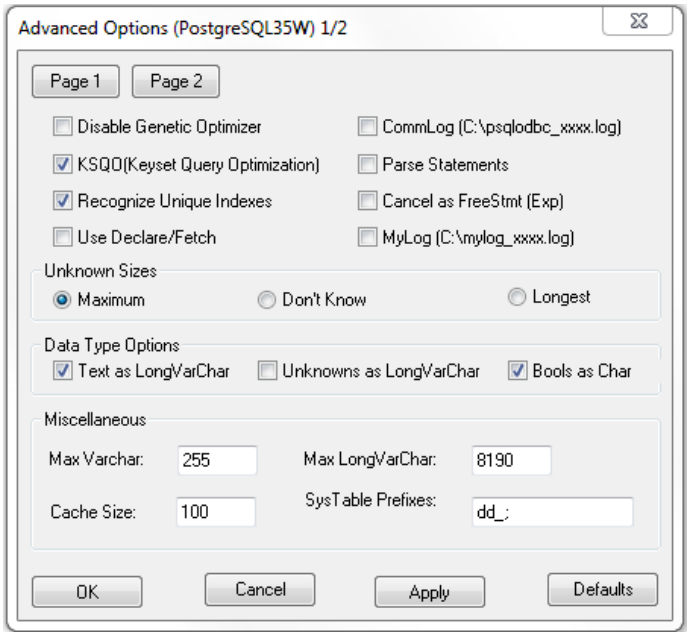
11. Complete the data source information fields:

Data Source	The data source name (DSN). Media Server uses this string to connect to the database server.

Database	The name of the database that you created in Step 2.
Server	The IP address or hostname of the server that the database server is installed on.
User Name	The user name to connect to the database server with.
Description	An optional description for the data source.
SSL Mode	Whether to use SSL to connect to the database server. **Note:** To enable SSL mode, you must also configure the database server to support SSL. For instructions, refer to the PostgreSQL documentation.
Port	The port to use to communicate with the database server.
Password	The password for the user account that connects to the database server.

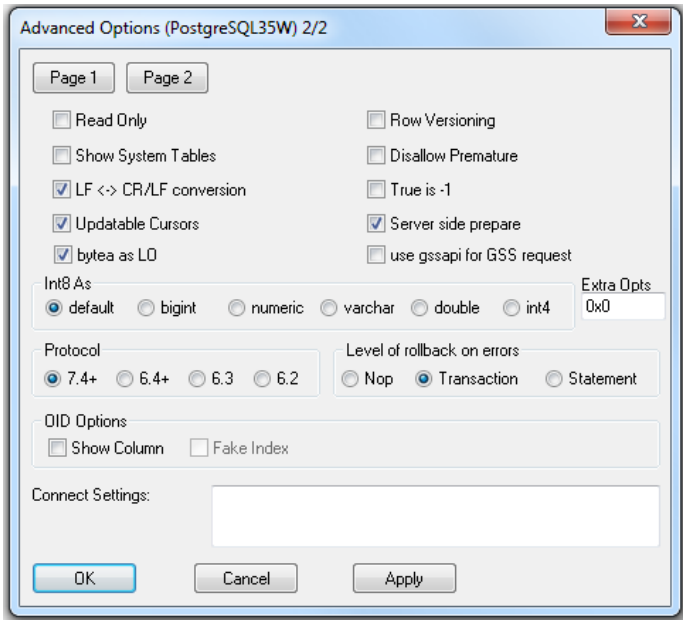
12. Click **Datasource**.

    The Advanced Options (*driverName*) 1/2 dialog box opens.



13. (Optional) HPE recommends that you select the **Use Declare/Fetch** check box, to reduce memory use.
14. Click **Page 2**.

    The Advanced Options (*driverName*) 2/2 dialog box opens.



15. Select the **bytea as LO** check box.
16. Click **Apply** and then **OK**.

    The Advanced Options (*driverName*) 2/2 dialog box closes.
17. In the PostgreSQL Unicode ODBC Driver (psqlODBC) Setup dialog box, click **Test** to test the connection.

    The Connection Test box opens containing a message describing whether the connection was successful. If the connection failed, use the information in the message to resolve any issues.
18. Click **OK** to close the Connection Test box.
19. In the PostgreSQL Unicode ODBC Driver (psqlODBC) Setup dialog box, click **Save** to close the dialog box.
20. In the ODBC Data Source Administrator dialog box, click **OK** to close the dialog box.
21. You can now configure Media Server to connect to the database (see "Configure Media Server" on page 44).

Set Up a PostgreSQL Database on Linux

To use Media Server with a PostgreSQL database, you must install a PostgreSQL server and ODBC driver, and configure Media Server to connect to the database through the driver.

The procedure describes how to set up a PostgreSQL database on a CentOS 6 distribution.

To set up a PostgreSQL Media Server database on Linux

1. Edit the .repo file to exclude PostgreSQL:
   a. Open the CentOS-Base.repo file with a text editor. The file is usually located in `/etc/yum.repos.d`.

b. Add the following line to the `[base]` and `[updates]` sections:

```
exclude=postgresql*
```

2. Download the PostgreSQL 9.*x* RPM file for your Linux distribution from the PostgreSQL Yum repository on www.postgresql.org. For example:

```
curl -O http://yum.postgresql.org/9.3/redhat/rhel-5-x86_64/pgdg-centos93-9.3-1.noarch.rpm
```

3. Install the PostgreSQL RPM file by running the command:

```
sudo rpm -i RPM
```

where *RPM* is the name of the downloaded RPM file.

4. Install the required packages from the RPM file. Ensure that these include the ODBC driver. For example:

```
sudo yum install postgresql93 postgresql93-odbc
```

5. Add the PostgreSQL bin directory path to the `PATH` environmental variable by running the command:

```
export PATH=$PATH:binDirectoryPath
```

Note: This step enables you to use the command `psql` to start the PostgreSQL command-line tool (psql) from the terminal. If the directory path is not added to the `PATH` variable, you must specify the psql.exe file path in the terminal to start psql.

6. Initialize and start PostgreSQL.
   a. Initialize the server by running the command:

```
sudo service postgresql-9.3 initdb
```

   b. Start the server by running the command:

```
sudo service postgresql-9.3 start
```

7. Log on to the psql command-line tool by running the command:

```
sudo -u postgres psql
```

8. Run a `CREATE DATABASE` command to create a new database. Specify the following database settings.

Database name	Any name.
Encoding	Must be Unicode–either UTF8 or UCS2.
Collation	Any that is compatible with the encoding.
Locale	Any that is compatible with the encoding.

For example:

```
CREATE DATABASE myDatabase WITH ENCODING 'UTF8' LC_COLLATE='en_US.UTF-8' LC_CTYPE='en_US.UTF-8';
```

9. Connect to the new database using the command:

   ```
   \c databaseName
   ```

10. Run the postgres.sql script provided in the Media Server installation directory. This script sets up the database schema that Media Server requires. The schema is inserted inside the `public` schema.
    a. HPE recommends running the following command to ensure that the script stops running if it encounters an error:

       ```
       \set ON_ERROR_STOP on
       ```

    b. Run the script using the command:

       ```
       \i 'path/postgres.sql'
       ```

       where *path* is the script file path.
11. Grant privileges to the user that Media Server will connect as. Required privileges are:

Database	Create Temporary Tables
All tables	Select, Insert, Update, Delete
All functions and stored procedures	Execute

    If security is not a consideration, grant all privileges.

    For example:

    ```
    GRANT TEMP ON DATABASE databaseName TO userName;
    ```

    ```
    GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO
    userName;
    ```

    ```
    GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO userName;
    ```

    where,

databaseName	is the name of the database that you created.
userName	is the user name that Media Server will connect as.

12. Install unixODBC driver manager version 2.2.14 or later. If using the Yum package manager, run the command:

    ```
    sudo yum install unixODBC
    ```

13. Configure the data source.
    a. Open the `odbc.ini` file with a text editor. This file is usually stored in the `/etc` directory.
    b. Add the data source name in square brackets. The name can be any string. For example:

       ```
       [PostgreSQL_1]
       ```

    c. Under the data source name, set the following parameters.

Parameter	Description

`Driver`	The driver to use.
`ServerName`	The IP address or hostname of the server that the database server is installed on.
`Port`	The port to use to communicate with the database server.
`UserName`	The user name to connect to the database server with.
`Password`	The password for the user account that connects to the database server.
`Database`	The name of the database that you created in Step 2.
`ByteAsLongVarBinary`	You must set this parameter to `1`. **Caution:** If this value is not set to `1`, Media Server fails to start.
`UseDeclareFetch`	(Optional) HPE recommends setting this parameter to `1`, to reduce memory use.

For example:

```
[PostgreSQL_1]
Driver=PostgreSQL
ServerName=localhost
Port=5432
UserName=postgres
Password=password
Database=myDatabase
ByteAsLongVarBinary=1
UseDeclareFetch=1
```

Note: You can set other parameters in this file, but these have not been tested with Media Server.

d. Save and close the file.

14. Configure the ODBC driver.
    a. Open the odbcinst.ini file with a text editor. This file is usually stored in the `/etc` directory.
    b. If not already present, add the database server name in square brackets. For example:

    ```
    [PostgreSQL]
    ```

    c. Under the database server name, set the following parameters.

Parameter	Description
Description	A description of the driver instance.
Driver64	The location of the PostgreSQL driver library file.

Setup64	The location of the driver installer file.
FileUsage	Set this parameter to `1`.

For example:

```
[PostgreSQL]
Description=ODBC for PostgreSQL
Driver64=/usr/pgsql-9.3/lib/psqlodbc.so
Setup64=/usr/lib64/libodbcpsqlS.so
FileUsage=1
```

Note: You can set other parameters in this file, but these have not been tested with Media Server.

d. Save and close the file.

15. You can now configure Media Server to connect to the database (see "Configure Media Server" on page 44).

Set Up a MySQL Database on Windows

To use Media Server with a MySQL database, you must download and install a MySQL server and ODBC driver, and configure Media Server to connect to the database through the driver.

To set up a MySQL Media Server database on Windows

1. Download and install a MySQL server and MySQL Connector/ODBC (which contains the Unicode driver). During installation, set up a user account with superuser privileges. For instructions, refer to the MySQL documentation on www.mysql.com.

   Note: Once installed, the MySQL server appears in the Services tab in Windows Task Manager.

2. Configure the database server for use with Media Server:
   a. Open the configuration or options file for the MySQL server (usually named `my.ini`).
   b. So that Media Server can send large amounts of binary data (images) to the database, set the configuration parameter **`max_allowed_packet=1073741824`**.
   c. Save and close the configuration file.
3. Add the MySQL `bin` directory path to the `PATH` environmental variable.

   Note: This step enables you to use the command `mysql` to start the mysql command-line tool from the Windows Command Prompt. If the directory path is not added to the `PATH` variable, you must specify the mysql.exe file path in the Command Prompt to start psql.

4. Open the mysql command line tool:
   a. In the Windows Command Prompt, run the command:

   ```
   mysql -u userName -p
   ```

   b. Enter your password when prompted.

5. Run a `CREATE DATABASE` command to create a new database. Specify the following database settings.

Database name	Any name.
Character set	Must be Unicode–either UTF8 or UCS2.
Collation	Any that is compatible with the encoding.

   For example:

```
CREATE DATABASE myDatabase CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

6. Run the `my.sql` script provided in the Media Server installation directory. This script sets up the database schema that Media Server requires.
   a. Close the mysql command-line tool:

```
quit
```

   b. In the Windows Command Prompt, run the following command:

```
mysql -u userName -p -v -D databaseName -e "source path/my.sql"
```

   where,

userName	is the MySQL user name.
databaseName	is the name of the database you created in Step 2.
path	is the path to the my.sql file.

   Note: Running the script non-interactively from the terminal ensures that the script terminates if an error occurs.

   c. Enter your password when prompted.
7. Grant privileges to the user that Media Server will connect as. Required privileges are:

Database	Create Temporary Tables
All tables	Select, Insert, Update, Delete
All functions and stored procedures	Execute

   If security is not a consideration, grant all privileges.
   a. Start the mysql command-line tool:

```
mysql
```

   b. Run the `GRANT` commands:

```
GRANT CREATE TEMPORARY TABLES ON databaseName.* TO userName;
GRANT SELECT, INSERT, UPDATE, DELETE ON databaseName.* TO userName;
GRANT EXECUTE ON databaseName.* TO userName;
```

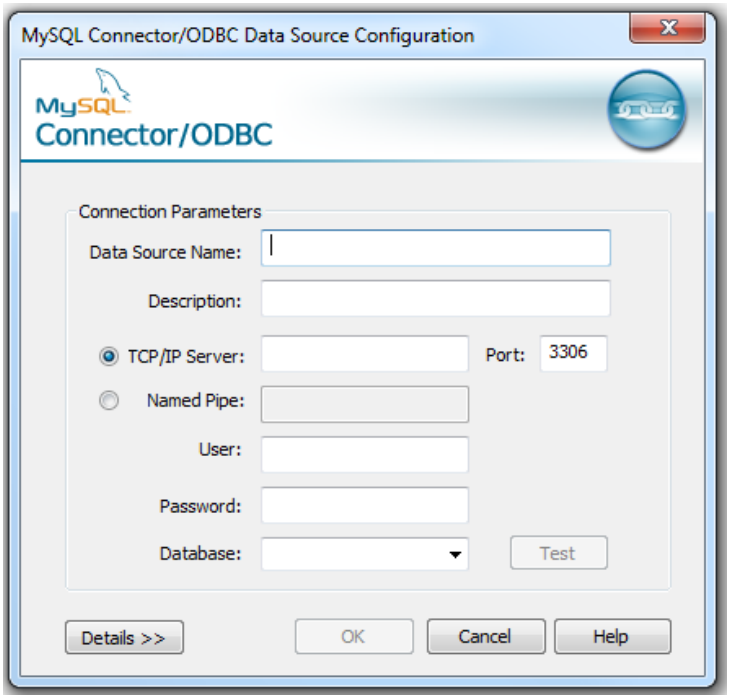
   where,

databaseName	is the name of the database you created in Step 2.
userName	is the user name that Media Server will connect as.

c. Close the mysql command-line tool:

```
quit
```

8. Open the Data Sources (ODBC) program:
   a. In the Windows Control Panel, click **System and Security**.
      The System and Security window opens.
   b. Click **Administrative Tools**.
      The Administrative Tools window opens.
   c. Double-click **Data Sources (ODBC)**.
      The ODBC Data Source Administrator dialog box opens.
9. In the User DSN tab, click **Add...** .
   The Create New Data Source dialog box opens.
10. Select the MySQL ODBC Unicode driver from the list and click **Finish**.
    The MySQL Connector/ODBC Data Source Configuration dialog box opens.



11. Complete the Connection Parameters fields:

Data Source Name	The data source name (DSN). Choose any string. Media Server can use this string to connect to the database server.
Description	An optional description for the data source.

TCP/IP Server	The IP address or hostname of the server that the database server is installed on.
Port	The port to use to communicate with the database server.
User	The user name to connect to the database server with.
Password	The password for the user account that connects to the database server.
Database	The name of the database that you created in Step 2.

12. Click **Test** to test the connection.

    The Connection Test box opens containing a message describing whether the connection was successful. If the connection failed, use the information in the message to resolve any issues.
13. Click **OK** to close the Connection Test box.
14. In the MySQL Connector/ODBC Data Source Configuration dialog box, click **OK** to close the dialog box.
15. In the ODBC Data Source Administrator dialog box, click **OK** to close the dialog box.
16. You can now configure Media Server to connect to the database (see "Configure Media Server" on page 44).

Set Up a MySQL Database on Linux

To use Media Server with a MySQL database, you must install a MySQL server and ODBC driver, and configure Media Server to connect to the database through the driver.

The procedure describes how to set up a MySQL database on a CentOS 6 distribution.

To set up a MySQL Media Server database on Linux

1. Install a MySQL server. (Ensure that the package includes the mysql command-line tool.) For instructions, refer to the MySQL documentation on www.mysql.com.
2. Configure the database server for use with Media Server:
   a. Open the configuration or options file for the MySQL server (usually named `my.ini`).
   b. So that Media Server can send large amounts of binary data (images) to the database, set the configuration parameter **`max_allowed_packet=1073741824`**.
   c. Save and close the configuration file.
3. Add the MySQL bin directory path to the `PATH` environmental variable by running the command:

   ```
   export PATH=$PATH:binDirectoryPath
   ```

   Note: This step enables you to use the command `mysql` to start the mysql command-line tool from the terminal. If the directory path is not added to the `PATH` variable, you must specify the mysql.exe file path in the terminal to start mysql.
4. Start the mysql command-line tool. In the terminal, run the command:

   ```
   mysql
   ```
5. Run a `CREATE DATABASE` command to create a new database. Specify the following database

settings.

Database name	Any name.
Character set	Must be Unicode–either UTF8 or UCS2.
Collation	Any that is compatible with the encoding.

For example:

```
CREATE DATABASE myDatabase CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

6. Run the my.sql script provided in the Media Server installation directory. This script sets up the database schema that Media Server requires.
   a. Close the mysql command-line tool:

      ```
      quit
      ```

   b. In the terminal, run the command:

      ```
      mysql -u userName -p -v -D databaseName -e "source path/my.sql"
      ```

      where,

userName	is the MySQL user name.
databaseName	is the name of the database you created in Step 3.
path	is the script file path.

      Note: Running the script non-interactively from the terminal ensures that the script terminates if an error occurs.

7. Grant privileges to the user that Media Server will connect to the MySQL server as. Required privileges are:

Database	Create Temporary Tables
All tables	Select, Insert, Update, Delete
All functions and stored procedures	Execute

If security is not a consideration, grant all privileges.

   a. Start the mysql command-line tool:

      ```
      mysql
      ```

   b. Run the `GRANT` commands:

      ```
      GRANT CREATE TEMPORARY TABLES ON databaseName.* TO userName;
      GRANT SELECT, INSERT, UPDATE, DELETE ON databaseName.* TO username;
      GRANT EXECUTE ON databaseName.* TO username;
      ```

      where,

databaseName	is the name of the database you created in Step 2.

userName is the user name that Media Server will connect as.

c. Close the mysql command-line tool:

```
quit
```

8. Install unixODBC driver manager version 2.2.14 or later. If you have the relevant Yum repository, you can run the command in the terminal:

```
sudo yum install unixODBC
```

9. Install the MySQL driver. If you have the relevant Yum repository, you can run the command in the terminal:

```
sudo yum install mysql-connector-odbc
```

10. Configure the data source.
    a. Open the `odbc.ini` file with a text editor. This file is usually stored in the `/etc` directory.
    b. Add the data source name in square brackets. The name can be any string. For example:

```
[MySQL_1]
```

    c. Under the data source name, set the following parameters.

Parameter	Description
`Driver`	The driver to use.
`Server`	The IP address or hostname of the server that the database server is installed on.
`Port`	The port to use to communicate with the database server.
`User`	The user name to connect to the database server with.
`Password`	The password for the user account that connects to the database server.
`Database`	The name of the database that you created in Step 3.

For example:

```
[MySQL_1]
Driver=MySQL
Server=localhost
Port=5432
User=mysql
Password=password
Database=myDatabase
```

Note: You can set other parameters in this file, but these have not been tested with Media Server.

    d. Save and close the file.

11. Configure the ODBC driver.

a. Open the odbcinst.ini file with a text editor. This file is usually stored in the `/etc` directory.
b. If not already present, add the database server name in square brackets. For example:

```
[MySQL]
```

a. Set the following parameters.

Parameter	Description
Description	A description of the driver instance.
Driver64	The location of the MySQL driver library file.
Setup64	The location of the driver installer file.
FileUsage	Set this parameter to **1**.

For example:

```
[MySQL]
Description=ODBC for MySQL
Driver64=/usr/lib64/libmyodbc5.so
Setup64=/usr/lib64/libodbcmyS.so
FileUsage=1
```

Note: You can set other parameters in this file, but these have not been tested with Media Server.

b. Save and close the file.

12. You can now configure Media Server to connect to the database (see "Configure Media Server" below).

Configure Media Server

To configure Media Server to use an external database

1. Stop Media Server, if it is running.
2. Open the Media Server configuration file (`mediaserver.cfg`) with a text editor.
3. Find the `[Database]` section of the configuration file. Create this section if it does not exist.
4. Set the following parameters:

`DatabaseType`	The type of database to use, either **`mysql`** or **`postgres`**.

`ODBCConnectionString`	The ODBC Connection string to use to connect to the database. For example:
	`DSN=MyDatabase;`
	`Driver = {PostgreSQL UNICODE}; Server = IPAddress; Port = port; Database = myDatabase; Uid = myUsername; Pwd = myPassword;`

```
Driver = {MySQL ODBC 5.x UNICODE Driver}; Server =
IPAddress; Database = myDatabase; User = myUsername;
Password = myPassword; Option = 3;
```

For example:

```
[Database]
DatabaseType=postgres
ODBCConnectionString=DSN=MyDatabase;
```

5. If you are running Media Server on Linux, set the following parameter:

`ODBCDriverManager`	The unixODBC Driver Manager shared object file.

For example:

```
ODBCDriverManager=libodbc.so
```

6. Save and close the configuration file.
7. Start Media Server.

Upgrade Media Server

If you have previously installed Media Server and need to upgrade to the latest version, follow these steps.

To upgrade to the latest version of Media Server

1. Make a backup of the following files from your current installation:
   - The Media Server configuration file, `mediaserver.cfg`.
   - Any session configuration files, from the `configurations` folder.
   - Any scene analysis training data, from the scene analysis training directory.
   - If you are using an internal database, the Media Server database which contains your training data. By default, the database is named `mediaserver.db`.
2. Perform a clean installation of the latest version of Media Server.
3. Check the HPE Autonomy Customer Support Site and install the latest patch, if one has been released:
   a. Download the latest patch from the HPE Autonomy Customer Support Site. Patches are cumulative, so you only need to download the latest patch.
   b. Unzip the files into the Media Server installation directory, overwriting any files that already exist.
4. Copy your configuration files into the new installation, overwriting the configuration files that were installed.
5. Restore or upgrade the database:

- If you are using an internal database, copy the database file into the new installation.
- If you are using an external database, you might need to run a script to upgrade the database schema. For more information, see "Upgrade the Database Schema" below.

Upgrade the Database Schema

Sometimes the schema of the Media Server database must change in order to provide new features or enhancements. If you are using an internal database, any schema changes are applied automatically. If you are using a database that is hosted on an external database server, you must run an upgrade script when you upgrade Media Server.

Note: Media Server versions 11.1 and later do not include upgrade scripts to upgrade from database schema used by Media Server 10.x. If you are using Media Server 10.x with either an internal or external database, upgrade to Media Server 11.0 first, and then upgrade to the latest version.

HPE provides scripts to upgrade to the latest version of the database schema from each of the earlier versions. The following table describes the schema changes for the Media Server database.

Schema version	Media Server version	Script to run to upgrade to latest schema
3	11.0.x	You are using the latest database schema
2	10.11.x	`mysql-upgrade_from_v2.sql` (for MYSQL databases) `postgres-upgrade_from_v2.sql` (for PostgreSQL databases)
1	10.10.x	`mysql-upgrade_from_v1.sql` (for MYSQL databases) `postgres-upgrade_from_v1.sql` (for PostgreSQL databases)

Running one of these scripts copies your training data and adds it to the database using the latest schema. These scripts do not remove training data stored using earlier schema versions. This means that you can continue to use the database with an earlier version of Media Server. Be aware that if you use multiple versions of Media Server, any new training you perform is only added to the database using the schema for the Media Server that performs the training. For example, if you upgrade from schema version 1 to schema version 2, you can perform training with Media Server 10.10.x and Media Server 10.11.x. However, any training you perform with Media Server 10.10.x is available only to Media Server version 10.10.x, and any training that you perform using Media Server 10.11.x is available only to Media Server version 10.11.x.

After a successful schema upgrade you can remove data stored using earlier schema versions. This saves storage space on the database server. HPE provides scripts to remove the data, named `mysql-purge_before_v`*X*`.sql` (for MySQL databases) or `postgres-purge_before_v`*X*`.sql` (for PostgreSQL databases), where *X* is the oldest schema version you want to retain in the database. For example, if you upgrade from schema version 1 to schema version 2, and do not want to use Media Server 10.10.x again, you can run `mysql-purge_before_v2.sql` or `postgres-purge_before_v2.sql` to remove schema version 1 from the database.

To upgrade the database schema

1. In the table above, find the version of Media Server that you are upgrading from.
2. Run the corresponding upgrade script for your database, using the same command syntax as used to create the database (see the following topics):
   - "Set Up a PostgreSQL Database on Windows" on page 30
   - "Set Up a PostgreSQL Database on Linux" on page 34
   - "Set Up a MySQL Database on Windows" on page 38
   - "Set Up a MySQL Database on Linux" on page 41

   Note: Run the upgrade script using the `psql` command-line tool (for PostgreSQL databases) or the `mysql` command-line tool (for MySQL databases). The script contains instructions that are only supported when the script runs through these tools.
3. Start Media Server 11.1, and run training and analysis as normal.
4. (Optional) When you have confirmed that the upgrade was successful, remove the training data stored using earlier schema versions by running the relevant script, either `mysql-purge_before_vX.sql`, or `postgres-purge_before_vX.sql`, where *X* is the oldest schema version you want to retain in the database.

Licenses

To use HPE IDOL solutions, you must have a running HPE License Server, and a valid license key file for the products that you want to use. Contact HPE Big Data Support to request a license file for your installation.

License Server controls the IDOL licenses, and assigns them to running components. License Server must run on a machine with a static, known IP address, MAC address, or host name. The license key file is tied to the IP address and ACI port of your License Server and cannot be transferred between machines. For more information about installing License Server and managing licenses, see the *License Server Administration Guide*.

When you start Media Server, it requests a license from the configured License Server. You must configure the host and port of your License Server in the Media Server configuration file.

You can revoke the license from a product at any time, for example, if you want to change the client IP address or reallocate the license.

Caution: Taking any of the following actions causes the licensed module to become inoperable.

You **must not**:

- Change the IP address of the machine on which a licensed module runs (if you use an IP address to lock your license).
- Change the service port of a module without first revoking the license.

- Replace the network card of a client without first revoking the license.
- Remove the contents of the `license` and `uid` directories.

All modules produce a `license.log` and a `service.log` file. If a product fails to start, check the contents of these files for common license errors. See "Troubleshoot License Errors" on page 50.

Display License Information

You can verify which modules you have licensed either by using the IDOL Admin interface, or by sending the `LicenseInfo` action from a web browser.

To display license information in IDOL Admin

- In the **Control** menu of the IDOL Admin interface for your License Server, click **Licenses**.

  The **Summary** tab displays summary information for each licensed component, including:

  - The component name.
  - The number of seats that the component is using.
  - The total number of available seats for the component.
  - (Content component only) The number of documents that are currently used across all instances of the component.
  - (Content component only) The maximum number of documents that you can have across all instances of the component.

  The **Seats** tab displays details of individual licensed seats, and allows you to revoke licenses.

To display license information by sending the LicenseInfo action

- Send the following action from a web browser to the running License Server.

`http://`*`LicenseServerHost`*`:`*`Port`*`/action=LicenseInfo`

where:

`LicenseServerHost`	is the IP address of the machine where License Server resides.
`Port`	is the ACI port of License Server (specified by the `Port` parameter in the `[Server]` section of the License Server configuration file).

In response, License Server returns the requested license information. This example describes a license to run four instances of IDOL Server.

```
<?xml version="1.0" encoding="UTF-8" ?>
<autnresponse xmlns:autn="http://schemas.autonomy.com/aci/">
  <action>LICENSEINFO</action>
  <response>SUCCESS</response>
  <responsedata>
    <LicenseDiSH>
```

```
      <LICENSEINFO>
        <autn:Product>
          <autn:ProductType>IDOLSERVER</autn:ProductType>
          <autn:TotalSeats>4</autn:TotalSeats>
          <autn:SeatsInUse>0</autn:SeatsInUse>
        </autn:Product>
      </LICENSEINFO>
    </LicenseDiSH>
  </responsedata>
</autnresponse>
```

Configure the License Server Host and Port

Media Server is licensed through HPE License Server. In the Media Server configuration file, specify the information required to connect to the License Server.

To specify the license server host and port

1. Open your configuration file in a text editor.
2. In the `[License]` section, modify the following parameters to point to your License Server.

   `LicenseServerHost` The host name or IP address of your License Server.

   `LicenseServerACIPort` The ACI port of your License Server.

   For example:

   ```
   [License]
   LicenseServerHost=licenses
   LicenseServerACIPort=20000
   ```

3. Save and close the configuration file.

Revoke a Client License

After you set up licensing, you can revoke licenses at any time, for example, if you want to change the client configuration or reallocate the license. The following procedure revokes the license from a component.

Note: If you cannot contact the client (for example, because the machine is inaccessible), you can free the license for reallocation by sending an `AdminRevokeClient` action to the License Server. For more information, see the *License Server Administration Guide*.

To revoke a license

1. Stop the HPE solution that uses the license.
2. At the command prompt, run the following command:

   `InstallDir/ExecutableName``[.exe] -revokelicense -configfile` *`cfgFilename`*

This command returns the license to the License Server.

You can send the `LicenseInfo` action from a web browser to the running License Server to check for free licenses. In this sample output from the action, one IDOL Server license is available for allocation to a client.

```
<autn:Product>
   <autn:ProductType>IDOLSERVER</autn:ProductType>
   <autn:Client>
      <autn:IP>192.123.51.23</autn:IP>
      <autn:ServicePort>1823</autn:ServicePort>
      <autn:IssueDate>1063192283</autn:IssueDate>
      <autn:IssueDateText>10/09/2003 12:11:23</autn:IssueDateText>
   </autn:Client>
      <autn:TotalSeats>2</autn:TotalSeats>
      <autn:SeatsInUse>1</autn:SeatsInUse>
</autn:Product>
```

Troubleshoot License Errors

The table contains explanations for typical licensing-related error messages.

License-related error messages

Error message	Explanation
Error: Failed to update license from the license server. Your license cache details do not match the current service configuration. Shutting the service down.	The configuration of the service has been altered. Verify that the service port and IP address have not changed since the service started.
Error: License for *ProductName* is invalid. Exiting.	The license returned from the License Server is invalid. Ensure that the license has not expired.
Error: Failed to connect to license server using cached licensed details.	Cannot communicate with the License Server. The product still runs for a limited period; however, you should verify whether your License Server is still available.
Error: Failed to connect to license server. Error code is SERVICE: *ErrorCode*	Failed to retrieve a license from the License Server or from the backup cache. Ensure that your License Server can be contacted.
Error: Failed to decrypt license keys. Please contact Autonomy support. Error code is SERVICE:*ErrorCode*	Provide HPE Big Data Support with the exact error message and your license file.
Error: Failed to update the license from the license server. Shutting down	Failed to retrieve a license from the License Server or from the backup cache. Ensure that your License Server can be contacted.
Error: Your license keys are invalid. Please contact Autonomy support. Error code is SERVICE:*ErrorCode*	Your license keys appear to be out of sync. Provide HPE Big Data Support with the exact error message and your license file.

License-related error messages, continued

Error message	Explanation
Failed to revoke license: No license to revoke from server.	The License Server cannot find a license to revoke.
Failed to revoke license from server *LicenseServer Host*:*LicenseServerPort*. Error code is *ErrorCode*	Failed to revoke a license from the License Server. Provide HPE Big Data Support with the exact error message.
Failed to revoke license from server. An instance of this application is already running. Please stop the other instance first.	You cannot revoke a license from a running service. Stop the service and try again.
Failed to revoke license. Error code is SERVICE:*ErrorCode*	Failed to revoke a license from the License Server. Provide HPE Big Data Support with the exact error message.
Your license keys are invalid. Please contact Autonomy Support. Error code is ACISERVER:*ErrorCode*	Failed to retrieve a license from the License Server. Provide HPE Big Data Support with the exact error message and your license file.
Your product ID does not match the generated ID.	Your installation appears to be out of sync. Forcibly revoke the license from the License Server and rename the `license` and `uid` directories.
Your product ID does not match this configuration.	The service port for the module or the IP address for the machine appears to have changed. Check your configuration file.

Distribute Media Server Operations

In large systems where you want to process large numbers of images, documents, and videos, you can install Media Server on more than one machine. In this case, you can use a Distributed Action Handler (DAH) to distribute actions to each Media Server, to ensure that each Media Server receives a similar number of requests.

You can use the IDOL Server installer to install the DAH. For more information about installing the DAH, refer to the *Distributed Action Handler Administration Guide*.

Chapter 3: Configure Media Server

This section describes how to configure Media Server.

The Media Server Configuration File

The configuration file is named `mediaserver.cfg`, and is located in the Media Server installation directory. You can modify the configuration file to customize the operation of Media Server.

The configuration file must include some parameters, such as those that specify the ports to use and those that configure the connection to the License Server.

The Media Server configuration file includes the following sections:

`[License]`	Contains parameters that configure the connection to your License Server.
`[Channels]`	Contains parameters that configure how many visual analysis, surveillance analysis, and audio analysis operations the Media Server can perform at one time. Media Server requests visual, surveillance, and audio channels from your License Server.
`[Logging]`	Contains parameters that determine how messages are logged. You can configure *log streams* to send different types of message to separate log files.
`[Paths]`	Contains parameters that specify the location of files required by Media Server.
`[Server]`	Contains general settings for Media Server. Specifies the ACI port of the Media Server and contains parameters that control the way the connector handles ACI requests.
`[Service]`	Contains settings that determine which machines can use and control the Media Server service.
`[Database]`	Contains settings required to connect to the database where training data is stored.

For a complete list of parameters that you can use in the configuration file, refer to the *Media Server Reference*.

Modify Configuration Parameter Values

You modify Media Server configuration parameters by directly editing the parameters in the configuration file. When you set configuration parameter values, you must use UTF-8.

Caution: You must stop and restart Media Server for new configuration settings to take effect.

This section describes how to enter parameter values in the configuration file.

Enter Boolean Values

The following settings for Boolean parameters are interchangeable:

```
TRUE = true = ON = on = Y = y = 1
```

```
FALSE = false = OFF = off = N = n = 0
```

Enter String Values

To enter a comma-separated list of strings when one of the strings contains a comma, you can indicate the start and the end of the string with quotation marks, for example:

ParameterName=**cat,dog,bird,"wing,beak",turtle**

Alternatively, you can escape the comma with a backslash:

ParameterName=**cat,dog,bird,wing\,beak,turtle**

If any string in a comma-separated list contains quotation marks, you must put this string into quotation marks and escape each quotation mark in the string by inserting a backslash before it. For example:

ParameterName=**"<font face=\"arial\" size=\"+1\"><b>","<p>"**

Here, quotation marks indicate the beginning and end of the string. All quotation marks that are contained in the string are escaped.

Include an External Configuration File

You can share configuration sections or parameters between ACI server configuration files. The following sections describe different ways to include content from an external configuration file.

You can include a configuration file in its entirety, specified configuration sections, or a single parameter.

When you include content from an external configuration file, the `GetConfig` and `ValidateConfig` actions operate on the combined configuration, after any external content is merged in.

In the procedures in the following sections, you can specify external configuration file locations by using absolute paths, relative paths, and network locations. For example:

```
../sharedconfig.cfg
K:\sharedconfig\sharedsettings.cfg
\\example.com\shared\idol.cfg
file://example.com/shared/idol.cfg
```

Relative paths are relative to the primary configuration file.

Note: You can use nested inclusions, for example, you can refer to a shared configuration file that references a third file. However, the external configuration files must not refer back to your original configuration file. These circular references result in an error, and Media Server does not start.

Similarly, you cannot use any of these methods to refer to a different section in your primary configuration file.

Include the Whole External Configuration File

This method allows you to import the whole external configuration file at a specified point in your configuration file.

To include the whole external configuration file

1. Open your configuration file in a text editor.
2. Find the place in the configuration file where you want to add the external configuration file.
3. On a new line, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. For example:

   ```
   < "K:\sharedconfig\sharedsettings.cfg"
   ```

4. Save and close the configuration file.

Include Sections of an External Configuration File

This method allows you to import one or more configuration sections from an external configuration file at a specified point in your configuration file. You can include a whole configuration section in this way, but the configuration section name in the external file must exactly match what you want to use in your file. If you want to use a configuration section from the external file with a different name, see "Merge a Section from an External Configuration File" on the next page.

To include sections of an external configuration file

1. Open your configuration file in a text editor.
2. Find the place in the configuration file where you want to add the external configuration file section.
3. On a new line, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file name, add the configuration section name that you want to include. For example:

   ```
   < "K:\sharedconfig\extrasettings.cfg" [License]
   ```

Note: You cannot include a section that already exists in your configuration file.

4. Save and close the configuration file.

Include a Parameter from an External Configuration File

This method allows you to import a parameter from an external configuration file at a specified point in your configuration file. You can include a section or a single parameter in this way, but the value in the external file must exactly match what you want to use in your file.

To include a parameter from an external configuration file

1. Open your configuration file in a text editor.
2. Find the place in the configuration file where you want to add the parameter from the external configuration file.
3. On a new line, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file name, add the name of the configuration section name that contains the parameter, followed by the parameter name. For example:

   ```
   < "license.cfg" [License] LicenseServerHost
   ```

   To specify a default value for the parameter, in case it does not exist in the external configuration file, specify the configuration section, parameter name, and then an equals sign (=) followed by the default value. For example:

   ```
   < "license.cfg" [License] LicenseServerHost=localhost
   ```

4. Save and close the configuration file.

Merge a Section from an External Configuration File

This method allows you to include a configuration section from an external configuration file as part of your Media Server configuration file. For example, you might want to specify a standard SSL configuration section in an external file and share it between several servers. You can use this method if the configuration section that you want to import has a different name to the one you want to use.

To merge a configuration section from an external configuration file

1. Open your configuration file in a text editor.
2. Find or create the configuration section that you want to include from an external file. For example:

   ```
   [SSLOptions1]
   ```

3. After the configuration section name, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. For example:

   ```
   [SSLOptions1] < "../sharedconfig/ssloptions.cfg"
   ```

If the configuration section name in the external configuration file does not match the name that you want to use in your configuration file, specify the section to import after the configuration file name. For example:

```
[SSLOptions1] < "../sharedconfig/ssloptions.cfg" [SharedSSLOptions]
```

In this example, Media Server uses the values in the `[SharedSSLOptions]` section of the external configuration file as the values in the `[SSLOptions1]` section of the Media Server configuration file.

Note: You can include additional configuration parameters in the section in your file. If these parameters also exist in the imported external configuration file, Media Server uses the values in the local configuration file. For example:

```
[SSLOptions1] < "ssloptions.cfg" [SharedSSLOptions]
SSLCACertificatesPath=C:\IDOL\HTTPConnector\CACERTS\
```

4. Save and close the configuration file.

Encrypt Passwords

HPE recommends that you encrypt all passwords that you enter into a configuration file.

Create a Key File

A key file is required to use AES encryption.

To create a new key file

1. Open a command-line window and change directory to the Media Server installation folder.
2. At the command line, type:

   ```
   autpassword -x -tAES -oKeyFile=./MyKeyFile.ky
   ```

   A new key file is created with the name `MyKeyFile.ky`

Caution: To keep your passwords secure, you must protect the key file. Set the permissions on the key file so that only authorized users and processes can read it. Media Server must be able to read the key file to decrypt passwords, so do not move or rename it.

Encrypt a Password

The following procedure describes how to encrypt a password.

To encrypt a password

1. Open a command-line window and change directory to the Media Server installation folder.
2. At the command line, type:

   ```
   autpassword -e -tEncryptionType [-oKeyFile] [-cFILE -sSECTION -pPARAMETER]
   PasswordString
   ```

where:

Option	Description
`-t EncryptionType`	The type of encryption to use: • **Basic** • **AES** For example: **-tAES** **Note:** AES is more secure than basic encryption.
`-oKeyFile`	AES encryption requires a key file. This option specifies the path and file name of a key file. The key file must contain 64 hexadecimal characters. For example: **-oKeyFile=./key.ky**
`-cFILE -sSECTION -pPARAMETER`	(Optional) You can use these options to write the password directly into a configuration file. You must specify all three options. • **-c**. The configuration file in which to write the encrypted password. • **-s**. The name of the section in the configuration file in which to write the password. • **-p**. The name of the parameter in which to write the encrypted password. For example: **-c./Config.cfg -sMyTask -pPassword**
PasswordString	The password to encrypt.

For example:

`autpassword -e -tBASIC MyPassword`

`autpassword -e -tAES -oKeyFile=./key.ky MyPassword`

`autpassword -e -tAES -oKeyFile=./key.ky -c./Config.cfg -sDefault -pPassword MyPassword`

The password is returned, or written to the configuration file.

Decrypt a Password

The following procedure describes how to decrypt a password.

To decrypt a password

1. Open a command-line window and change directory to the Media Server installation folder.
2. At the command line, type:

   `autpassword -d -tEncryptionType [-oKeyFile] PasswordString`

where:

Option	Description
`-t` *EncryptionType*	The type of encryption: • **Basic** • **AES** For example: **-tAES**
-oKeyFile	AES encryption and decryption requires a key file. This option specifies the path and file name of the key file used to decrypt the password. For example: **-oKeyFile=./key.ky**
PasswordString	The password to decrypt.

For example:

```
autpassword -d -tBASIC 9t3M3t7awt/J8A
```

```
autpassword -d -tAES -oKeyFile=./key.ky 9t3M3t7awt/J8A
```

The password is returned in plain text.

Specify Modules to Enable

You can disable Media Server modules to reduce the amount of memory used and the time required for the server to start.

By default, all modules included in your license are enabled. However, you can specify a list of modules to enable:

- **barcode**
- **demographics** - face demographics.
- **facedetect** - face detection.
- **facerecognize** - face recognition.
- **facestate** - provides information about facial expression, whether the person's eyes are open, and whether the person is wearing spectacles.
- **object**
- **objectclass**
- **ocr**

To specify the modules to enable

1. Open the Media Server configuration file in a text editor.
2. Find the `[Modules]` configuration section. If the section does not exist, add it by typing `[Modules]` on a new line.
3. Add the `Enable` parameter and specify a comma-separated list of modules to enable. For

example:

```
Enable=barcode,ocr,object
```

4. Save and close the configuration file.

   When you restart Media Server, only the specified modules are enabled.

Customize Logging

You can customize logging by setting up your own *log streams*. Each log stream creates a separate log file in which specific log message types (for example, action, index, application, or import) are logged.

To set up log streams

1. Open the Media Server configuration file in a text editor.
2. Find the `[Logging]` section. If the configuration file does not contain a `[Logging]` section, add one.
3. In the `[Logging]` section, create a list of the log streams that you want to set up, in the format *N=LogStreamName*. List the log streams in consecutive order, starting from 0 (zero). For example:

   ```
   [Logging]
   LogLevel=FULL
   LogDirectory=logs
   0=ApplicationLogStream
   1=ActionLogStream
   ```

   You can also use the `[Logging]` section to configure any default values for logging configuration parameters, such as `LogLevel`. For more information, see the *Media Server Reference*.
4. Create a new section for each of the log streams. Each section must have the same name as the log stream. For example:

   ```
   [ApplicationLogStream]
   [ActionLogStream]
   ```

5. Specify the settings for each log stream in the appropriate section. You can specify the type of logging to perform (for example, full logging), whether to display log messages on the console, the maximum size of log files, and so on. For example:

   ```
   [ApplicationLogStream]
   LogTypeCSVs=application
   LogFile=application.log
   LogHistorySize=50
   LogTime=True
   LogEcho=False
   LogMaxSizeKBs=1024

   [ActionLogStream]
   LogTypeCSVs=action
   LogFile=logs/action.log
   LogHistorySize=50
   LogTime=True
   ```

```
LogEcho=False
LogMaxSizeKBs=1024
```

6. Save and close the configuration file. Restart the service for your changes to take effect.

Example Configuration File

```
[License]
LicenseServerHost=localhost
LicenseServerACIPort=20000

[Server]
Port=14000
AdminClients=::1,127.0.0.1
QueryClients=*.*.*.*
Threads=4

[Service]
ServicePort=14001
ServiceStatusClients=*.*.*.*
ServiceControlClients=::1,127.0.0.1
Access-Control-Allow-Origin=*
UseEventLog=false

[Paths]
StaticDataDirectory=.
DatabasePath=mediaserver.db
FFmpegDirectory=./libav/
IsasTrainingDirectory=./ISAS/ISASTrainingCfgs/
IsasAlarmDirectory=ISAS/Alarms/
RollingBufferConfigPath=./encoding/rollingBuffer/rollingBuffer.cfg
ConfigDirectory=./configurations/
ActivityFile=./mediaserver-activity.html

[Resources]
SpeechToTextServers=localhost:15000
SpeakerIdServers=localhost:15000
IDOLServer=localhost:9000/News

[Modules]
Enable=barcode,faceanalyze,facedetect,facerecognize,object,objectclass,ocr

[Channels]
VisualChannels=1
SurveillanceChannels=1
AudioChannels=1

[Database]
```

```
DatabaseType=internal

[Process]
MaximumThreads=1

[Logging]
LogLevel=NORMAL
LogHistorySize=0
LogTime=true
LogEcho=true
LogMaxSizeKBs=20480
LogExpireAction=compress
LogMaxOldFiles=100
0=APP_LOG_STREAM
1=ACT_LOG_STREAM
2=ENGINE_LOG_STREAM
3=LIBAV_LOG_STREAM

[APP_LOG_STREAM]
LogFile=application.log
LogTypeCSVs=application

[ACT_LOG_STREAM]
LogFile=action.log
LogTypeCSVs=action

[ENGINE_LOG_STREAM]
LogFile=engine.log
LogTypeCSVs=engine

[LIBAV_LOG_STREAM]
LogLevel=ERROR
LogEcho=false
LogFile=libav.log
LogTypeCSVs=libav
```

Validate the Configuration File

You can use the `ValidateConfig` service action to check for errors in the configuration file.

Note: For the `ValidateConfig` action to validate a configuration section, Media Server must have previously read that configuration. In some cases, the configuration might be read when a task is run, rather than when the component starts up. In these cases, `ValidateConfig` reports any unread sections of the configuration file as unused.

To validate the configuration file

- Send the following action to Media Server:

  `http://Host:ServicePort/action=ValidateConfig`

  where:

Host	is the host name or IP address of the machine where Media Server is installed.
ServicePort	is the service port, as specified in the `[Service]` section of the configuration file.

Chapter 4: Start and Stop Media Server

The following sections describe how to start and stop Media Server.

Start Media Server

Note: Your License Server must be running before you start Media Server.

To start Media Server

- Start Media Server from the command line using the following command:

  ```
  mediaserver.exe -configfile configname.cfg
  ```

  where the optional `-configfile` argument specifies the path of a configuration file that you want to use.
- On Windows, if you have installed Media Server as a service, start the service from the Windows Services dialog box.
- On UNIX, if you have installed Media Server as a service, use one of the following commands:
  - On machines that use systemd:

    ```
    systemctl start mediaserver
    ```

  - On machines that use system V:

    ```
    service mediaserver start
    ```

Tip: On both Windows and UNIX platforms, you can configure services to start automatically when you start the machine.

Stop Media Server

You can stop Media Server by using one of the following procedures.

To stop Media Server

- Send the `Stop` service action to the service port.

  ```
  http://host:ServicePort/action=stop
  ```

where:

host	is the host name or IP address of the machine where Media Server is installed.
ServicePort	is the Media Server service port (specified in the [Service] section of the configuration file).

- On Windows platforms, if Media Server is running as a service, stop Media Server from the Windows Services dialog box.
- On UNIX platforms, if Media Server is running as a service, use one of the following commands:
  - On machines that use systemd:

    ```
    systemctl stop mediaserver
    ```

  - On machines that use system V:

    ```
    service mediaserver stop
    ```

Verify that Media Server is Running

After starting Media Server, you can run the following actions to verify that Media Server is running.

- GetStatus
- GetLicenseInfo

GetStatus

You can use the GetStatus service action to verify the Media Server is running. For example:

```
http://Host:ServicePort/action=GetStatus
```

Note: You can send the GetStatus action to the ACI port instead of the service port. The GetStatus ACI action returns information about the Media Server setup.

GetLicenseInfo

You can send a GetLicenseInfo action to Media Server to return information about your license. This action checks whether your license is valid and returns the operations that your license includes.

Send the GetLicenseInfo action to the Media Server ACI port. For example:

```
http://Host:ACIport/action=GetLicenseInfo
```

The following result indicates that your license is valid.

```
<autn:license>
       <autn:validlicense>true</autn:validlicense>
</autn:license>
```

As an alternative to submitting the GetLicenseInfo action, you can view information about your license, and about licensed and unlicensed actions, on the **License** tab in the Status section of IDOL Admin.

Access IDOL Admin

IDOL Admin is a utility that you can use to help monitor and configure Media Server. You can access IDOL Admin from a Web browser. For more information about IDOL Admin, refer to the *IDOL Admin User Guide*.

To access IDOL Admin

- Type the following URL into the address bar of your Web browser:

  `http://host:port/action=admin`

  where:

  host is the name or IP address of the host that Media Server is installed on.

  port is the Media Server ACI port.

Display Online Help

You can display the Media Server Reference by sending an action from your web browser. The Media Server Reference describes the actions and configuration parameters that you can use with Media Server.

For Media Server to display help, the help data file (`help.dat`) must be available in the installation folder.

To display help for Media Server

1. Start Media Server.
2. Send the following action from your web browser:

   `http://host:port/action=Help`

   where:

   host is the IP address or name of the machine on which Media Server is installed.

   port is the ACI port by which you send actions to Media Server (set by the `Port` parameter in the `[Server]` section of the configuration file).

   For example:

   `http://12.3.4.56:9000/action=help`

Chapter 5: Send Actions to Media Server

This section describes how to send actions to Media Server.

Synchronous and Asynchronous Actions

The actions that you send to Media Server can be *synchronous* or *asynchronous*.

Media Server does not respond to a synchronous action until it has completed the request. The result of the action is usually in the response to the request.

Media Server responds to an asynchronous action immediately. This means that you do not have to wait for a response if the action takes a long time. The response to an asynchronous action includes only a token. You can use this token to determine the status of the task through the `QueueInfo` action at a later time. An asynchronous request is added to a queue (each action has its own queue) and if the server is under load it is held in the queue until Media Server is ready to process it. The action queues are stored in a database, so requests are not lost if there is an interruption in service. You can also set priorities for asynchronous requests, so that the most important are processed first.

As a result, actions that complete quickly usually run synchronously. For example, `action=getstatus` is a synchronous action. Actions that can take a significant time to complete are usually asynchronous. These actions are asynchronous because otherwise requests might time out before Media Server is able to process them.

Send Actions to Media Server

You can make requests to Media Server by using either GET or POST HTTP request methods.

- A GET request sends parameter-value pairs in the request URL. GET requests are appropriate for sending actions that retrieve information from Media Server, such as the `GetStatus` action. For more information, see "Send Actions by Using a GET Method" on the next page.
- A POST request sends parameter-value pairs in the HTTP message body of the request. POST requests are appropriate for sending data to Media Server. In particular, you must use POST requests to upload and send files to Media Server. For more information, see "Send Data by Using a POST Method" on the next page.

HPE recommends that you send video data to Media Server by providing a URL or a path to a file. Video files are very large and sending them over HTTP, as either multipart/form-data or a base64 encoded string, could cause Media Server to slow and potentially lead to interruptions in service. Image, audio, and text files are generally much smaller and so you can send these files over HTTP.

Note: Media Server accepts HTTP strings that contain a maximum of 64,000 characters, and uploaded files with a maximum size of 10,000,000 bytes. You can override the default settings by setting the `MaxInputString` and `MaxFileUploadSize` configuration parameters in the `[Server]` section of the configuration file.

Send Actions by Using a GET Method

You can use GET requests to send actions that retrieve information from Media Server.

When you send an action using a GET method, you use a URL of the form:

`http://`*`host`*`:`*`port`*`/?action=`*`action`*`&`*`parameters`*

where:

`host`	is the IP address or name of the machine where Media Server is installed.
`port`	is the Media Server ACI port.
`action`	is the name of the action that you want to run. The `action` (or `a`) parameter must be the first parameter in the URL string, directly after the host and port details.
`parameters`	are the required and optional parameters for the action. These parameters can follow the `action` parameter in any order.

You must:

- Separate each parameter from its value with an equals symbol (`=`).
- Separate multiple values with a comma (`,`).
- Separate each parameter-value pair with an ampersand (`&`).

For more information about the actions that you can use with Media Server, refer to the *Media Server Reference*.

GET requests can send only limited amounts of data and cannot send files directly. However, you can set a parameter to a file path if the file is on a file system that Media Server can access. Media Server must also be able to read the file.

Send Data by Using a POST Method

You can send files and binary data directly to Media Server using a POST request. One possible way to send a POST request over a socket to Media Server is using the cURL command-line tool.

The data that you send in a POST request must adhere to specific formatting requirements. You can send only the following content types in a POST request to Media Server:

- `application/x-www-form-urlencoded`
- `multipart/form-data`

Tip: Media Server rejects POST requests larger than the size specified by the configuration parameter `MaxFileUploadSize`.

Application/x-www-form-urlencoded

The `application/x-www-form-urlencoded` content type describes form data that is sent in a single block in the HTTP message body. Unlike the query part of the URL in a GET request, the length of the data is unrestricted. However, Media Server rejects requests that exceed the size specified by the configuration parameter `MaxFileUploadSize`.

This content type is inefficient for sending large quantities of binary data or text containing non-ASCII characters, and does not allow you to upload files. For these purposes, HPE recommends sending data as `multipart/form-data` (see "Multipart/form-data" on the next page).

In the request:

- Separate each parameter from its value with an equals symbol (=).
- Separate multiple values with a comma (,).
- Separate each parameter-value pair with an ampersand (&).
- Base-64 encode any binary data.
- URL encode all non-alphanumeric characters, including those in base-64 encoded data.

Example

The following example base-64 encodes the file `image.jpg` into a file `imagedata.dat` and sends it (using cURL) as `application/x-www-form-urlencoded` data to Media Server located on the `localhost`, using port `14000`. The example action adds the image to a new face in an existing face database named `politicians`.

1. Base-64 encode the image.
   - On Windows, create a Powershell script called `base64encode.ps1` that contains the following:

     ```
     Param([String]$path)
     [convert]::ToBase64String((get-content $path -encoding byte))
     ```

     Then from the command line, run the script using Powershell:

     ```
     powershell.exe base64encode.ps1 image.jpg > imagedata.dat
     ```

   - On Linux:

     ```
     base64 -w 0 image.jpg > imagedata.dat
     ```

2. Send the request to Media Server:

   ```
   curl http://localhost:14000
      -d 'action=TrainFace&database=politicians&identifier=president'
      --data-urlencode imagedata@imagedata.dat
   ```

You can send multiple images in a single request by separating the binary data for each image by a comma (in `imagedata.dat`).

Multipart/form-data

In the `multipart/form-data` content type, the HTTP message body is divided into parts, each containing a discrete section of data.

Each message part requires a header containing information about the data in the part. Each part can contain a different content type; for example, `text/plain`, `image/png`, `image/gif`, or `multipart/mixed`. If a parameter specifies multiple files, you must specify the **`multipart/mixed`** content type in the part header.

Encoding is optional for each message part. The message part header must specify any encoding other than the default (7BIT).

Multipart/form-data is ideal for sending non-ASCII or binary data, and is the only content type that allows you to upload files. For more information about form data, see http://www.w3.org/TR/html401/interact/forms.html.

Note: In cURL, you specify each message part using the `-F` (or `--form`) option. To upload a file in a message part, prefix the file name with the `@` symbol. For more information on cURL syntax, see the cURL documentation.

Example

The following example sends `image.jpg` (using cURL) as `multipart/form-data` to Media Server located on the `localhost`, using port `14000`. The example action adds the image to a new face in an existing face database named `politicians`.

```
curl http://localhost:18000 -F action=TrainFace
                            -F database=politicians
                            -F identifier=president
                            -F imagedata=@image.jpg
```

Override Configuration Parameters

In actions that you send to Media Server, you can override some configuration parameters that are set in a configuration file.

To send new parameter values with an action, add the following to the action that you are sending:

`&[`*`ConfigSection`*`]`*`ParameterName=NewParameterValue`*

where:

`ConfigSection`	is the name of the configuration section that contains the parameter to override.
`ParameterName`	is the name of the configuration parameter to override.
`NewParameterValue`	is the new value to set.

For example, to override the `Orientation` configuration parameter for barcode analysis, in a single `process` action without changing the configuration file:

```
localhost:14000/action=process&configname=barcodes
                              &source=./media/document.pdf
                              &[BarcodeAnalysisTaskName]Orientation=Any
```

Use Asynchronous Actions

When you send an asynchronous action, Media Server adds the task to a queue and returns a token that you can use to check its status. Media Server performs the task when a thread becomes available.

Check the Status of an Asynchronous Action

To check the status of an asynchronous action, use the token that was returned by Media Server with the `QueueInfo` action. For more information about the `QueueInfo` action, see the *Media Server Reference*.

To check the status of an asynchronous action

- Send the `QueueInfo` action to Media Server with the following parameters.

`QueueName`	The name of the action queue that you want to check.
`QueueAction`	The action to perform. Set this parameter to **`GetStatus`**.
`Token`	(Optional) The token that the asynchronous action returned. If you do not specify a token, Media Server returns the status of every action in the queue.

For example:

```
/action=QueueInfo&QueueName=...&QueueAction=getstatus&Token=...
```

Cancel an Asynchronous Action that is Queued

To cancel an asynchronous action that is waiting in a queue, use the following procedure.

To cancel an asynchronous action that is queued

- Send the `QueueInfo` action to Media Server with the following parameters.

`QueueName`	The name of the action queue that contains the action to cancel.
`QueueAction`	The action to perform . Set this parameter to **`Cancel`**.
`Token`	The token that the asynchronous action returned.

Stop an Asynchronous Action that is Running

You can stop an asynchronous action at any point.

To stop an asynchronous action that is running

- Send the `QueueInfo` action to Media Server with the following parameters.

`QueueName`	The name of the action queue that contains the action to stop.
`QueueAction`	The action to perform. Set this parameter to **`Stop`**.
`Token`	The token that the asynchronous action returned.

Event Handlers

Some of the actions that you can send to Media Server are asynchronous. Asynchronous actions do not run immediately, but are added to a queue. This means that the person or application that sends the action does not receive an immediate response. However, you can configure Media Server to call an event handler when an asynchronous action starts, finishes, or encounters an error.

You might use an event handler to:

- Return data about an event back to the application that sent the action.
- Write event data to a text file, to log any errors that occur.

Media Server can call an event handler for the following events.

OnStart	The `OnStart` event is called when Media Server starts processing an asynchronous action from the queue.
OnFinish	The `OnFinish` event is called when Media Server successfully finishes processing an asynchronous action.
OnError	The `OnError` event is called when an asynchronous action fails and cannot continue.

Configure an Event Handler

Use the following procedure to configure an event handler.

To configure an event handler

1. Open the Media Server configuration file in a text editor.
2. Use the `OnStart`, `OnFinish`, or `OnError` parameter to specify the name of a section in the configuration file that contains event handler settings for the corresponding event.
   - To run an event handler for all asynchronous actions, set these parameters in the `[Actions]` section. For example:

     ```
     [Actions]
     OnStart=NormalEvents
     OnFinish=NormalEvents
     OnError=ErrorEvents
     ```

   - To run an event handler for a specific action, set these parameters in the `[`*`ActionName`*`]`

section, where *ActionName* is the name of the action. The following example runs an event handler when the `Process` action starts and finishes successfully:

```
[Process]
OnStart=NormalEvents
OnFinish=NormalEvents
```

3. Create a new section in the configuration file to contain the settings for your event handler. You must name the section using the name you specified with the `OnStart`, `OnFinish`, or `OnError` parameter.
4. In the new section, set the following parameters.

`LibraryName` (Required) The name of the library to use to handle the event.

- To write event data to a text file, set this parameter to **`TextFileHandler`**, and then set the `FilePath` parameter to specify the path of the file.
- To send event data to a URL, set this parameter to **`HttpHandler`**, and then use the HTTP event handler parameters to specify the URL, proxy server settings, credentials, and so on.

For example:

```
[NormalEvents]
LibraryName=TextFileHandler
FilePath=./events.txt

[ErrorEvents]
LibraryName=HTTPHandler
URL=http://handlers:8080/lo-proxy/callback.htm?
```

5. Save and close the configuration file. You must restart Media Server for your changes to take effect.

The following XML is an example of the ACI response sent to the event handler on the `OnFinish` event:

```
<action>
       <status>Finished</status>
       <queued_time>2009-Jun-08 17:57:29</queued_time>
       <time_in_queue>0</time_in_queue>
       <process_start_time>2012-Jun-08 17:57:29</process_start_time>
       <time_processing>86</time_processing>
       <token>MTkyLjE2OC45NS4yNDoxMTAwMDpJTkdFU1Q6LTEyOTc5NjgxODY=</token>
</action>
```

Process Multiple Requests Simultaneously

Media Server can process multiple requests simultaneously.

This means that if required, you can process multiple media sources concurrently. You can configure Media Server to do this to increase throughput (the total amount of media processed in a fixed time), or so that you can process several live video streams at the same time.

Process Asynchronous Requests Simultaneously

The way in which Media Server handles asynchronous actions is defined in the `[Actions]` section of the configuration file.

To configure the number of requests to process concurrently from each action queue, set the configuration parameter `MaximumThreads`. For example, to configure Media Server to simultaneously process up to two requests from each action queue set this parameter as follows:

```
[Actions]
MaximumThreads=2
```

You can override the value of `MaximumThreads` for specific actions by setting the parameter in the `[`*`ActionName`*`]` section of the configuration file (where *`ActionName`* is the name of the action). For example, if most of the requests received by your Media Server are for the `Process` action, you could increase the number of `Process` actions that run simultaneously by setting the `MaximumThreads` parameter in the `[Process]` section.

In the following example, Media Server runs up to four `Process` actions simultaneously, but for other asynchronous actions runs only one request at a time:

```
[Actions]
MaximumThreads=1

[Process]
MaximumThreads=4
```

Note: The `MaximumThreads` parameter specifies the number of actions that run simultaneously, not the number of threads that are used. For example, the number of threads required to run a `Process` action depends on the analysis, encoding, and other tasks that are configured.

Note: When they use Convolutional Neural Network (CNN) classifiers, multiple image classification tasks run in parallel only if they use different classifiers.

Process Synchronous Requests Simultaneously

To configure the number of synchronous requests to process simultaneously, set the `Threads` parameter in the `[Server]` section of the configuration file. In the following example, Media Server can process a total of eight synchronous requests simultaneously:

```
[Server]
Threads=8
```

Note: HPE recommends that you only run `Process` actions synchronously if you expect them to complete within a few seconds.

Store Action Queues in an External Database

Media Server provides asynchronous actions. Each asynchronous action has a queue to store requests until threads become available to process them. You can configure Media Server to store these queues either in an internal database file, or in an external database hosted on a database server.

The default configuration stores queues in an internal database. Using this type of database does not require any additional configuration.

You might want to store the action queues in an external database so that several servers can share the same queues. In this configuration, sending a request to any of the servers adds the request to the shared queue. Whenever a server is ready to start processing a new request, it takes the next request from the shared queue, runs the action, and adds the results of the action back to the shared database so that they can be retrieved by any of the servers. You can therefore distribute requests between components without configuring a Distributed Action Handler (DAH).

Note: You cannot use multiple servers to process a single request. Each request is processed by one server.

Note: Requests that contain multipart/form-data are always processed on the server that received them. These requests are added to the shared queue, but can only be taken from the queue by the server that received the request and placed it on the queue.

Tip: You can also store training data used by Media Server in an external database. You can use the same database server to store training data and asynchronous action queues. However, you must create separate databases for storing training data and action queues. As a result you must also create separate data source names (DSNs), and connection strings. For information about configuring Media Server to use an external database for training data, see "Configure Media Server" on page 44.

Prerequisites

- Supported databases:
  - PostgreSQL 9.0 or later.
  - MySQL 5.0 or later.

  Tip: These are the earliest versions of PostgreSQL and MySQL that you can use to store action queues. If you want to use the same database server to store training data, review the database requirements in the section "Set up a Database to Store Training Data" on page 29, because a later version might be required.
- If you use PostgreSQL, you must set the PostgreSQL ODBC driver setting `MaxVarChar` to `0` (zero). If you use a DSN, you can configure this parameter when you create the DSN. Otherwise, you can set the `MaxVarcharSize` parameter in the connection string.

Configure Media Server

To configure Media Server to use a shared action queue, follow these steps.

To store action queues in an external database

1. Stop Media Server, if it is running.
2. Open the Media Server configuration file.
3. Find the relevant section in the configuration file:
   - To store queues for all asynchronous actions in the external database, find the `[Actions]` section.
   - To store the queue for a single asynchronous action in the external database, find the section that configures that action.
4. Set the following configuration parameters.

`AsyncStoreLibraryDirectory`	The path of the directory that contains the library to use to connect to the database. Specify either an absolute path, or a path relative to the server executable file.
`AsyncStoreLibraryName`	The name of the library to use to connect to the database. You can omit the file extension. The following libraries are available: • **`postgresAsyncStoreLibrary`** - for connecting to a PostgreSQL database. • **`mysqlAsyncStoreLibrary`** - for connecting to a MySQL database.

`ConnectionString`	The connection string to use to connect to the database. The user that you specify must have permission to create tables in the database. For example:
	`ConnectionString=DSN=my_ASYNC_QUEUE`
	or
	`ConnectionString=Driver={PostgreSQL}; Server=10.0.0.1; Port=9876; Database=SharedActions; Uid=user; Pwd=password; MaxVarcharSize=0;`
	Tip: Do not use the same database, data source name, and connection string that you use for storing training data. You must create a separate database, DSN, and connection string.

For example:

```
[Actions]
AsyncStoreLibraryDirectory=acidlls
AsyncStoreLibraryName=postgresAsyncStoreLibrary
ConnectionString=DSN=ActionStore
```

5. If you are using the same database to store action queues for more than one type of component, set the following parameter in the `[Actions]` section of the configuration file.

`DatastoreSharingGroupName`	The group of components to share actions with. You can set this parameter to any string, but the value must be the same for each server in the group. For example, to configure several Media Servers to share their action queues, set this parameter to the same value in every Media Server configuration. HPE recommends setting this parameter to the name of the component. **Caution:** Do not configure different components (for example, two different types of connector) to share the same action queues. This will result in unexpected behavior.

```
For example:

[Actions]
...
DatastoreSharingGroupName=ComponentType
```

6. Save and close the configuration file.

   When you start Media Server it connects to the shared database.

Use XSL Templates to Transform Action Responses

You can transform the action responses returned by Media Server using XSL templates. You must write your own XSL templates and save them with either an `.xsl` or `.tmpl` file extension. For more information about XSL, see http://www.w3.org/TR/xslt.

After creating the templates, you must configure Media Server to use them, and then apply them to the relevant actions.

To enable XSL transformations

1. Ensure that the `autnxslt` library is located in the same directory as your configuration file. If the library is not included in your installation, you can obtain it from HPE Support.
2. Open the Media Server configuration file in a text editor.
3. In the `[Server]` section, set the `XSLTemplates` parameter to **`True`**.

   Caution: If `XSLTemplates` is set to **`True`** and the `autnxslt` library is not present in the same directory as the configuration file, the server will not start.

4. In the `[Paths]` section, set the `TemplateDirectory` parameter to the path to the directory that contains your XSL templates.
5. Save and close the configuration file.
6. Restart Media Server for your changes to take effect.

To apply a template to action output

- Add the following parameters to the action.

`Template`	The name of the template to use to transform the action output. Exclude the folder path and file extension.
`ForceTemplateRefresh`	(Optional) If you modified the template after the server started, set this parameter to **`True`** to force the ACI server to reload the template from disk rather than from the cache.

For example:

```
action=QueueInfo&QueueName=Process
                &QueueAction=GetStatus
                &Token=MTkyLjE2OC45NS4yNDox
                &Template=Form
```

In this example, Media Server applies the XSL template `Form.tmpl` to the response from a `QueueInfo` action.

Note: If the action returns an error response, Media Server does not apply the XSL template.

Chapter 6: Start Processing Media

This section describes how to create a configuration and use that configuration to start processing.

Configuration Overview

Before you begin processing a file or stream, you must create a configuration that instructs Media Server how to process the media.

You can configure Media Server by modifying the Media Server configuration file. However, you might want to process media from different sources, or run different analysis tasks for different types of media. Therefore HPE recommends that you do not configure ingestion, analysis, encoding, data transformation, event stream processing, or output in the Media Server configuration file. Instead, pass these settings to Media Server in the `process` action, when you start processing.

To pass a configuration to Media Server through the `process` action, you can:

- Create a configuration and save the file in the directory specified by the `ConfigDirectory` parameter, in the `[Paths]` section of the Media Server configuration file. When you run the `process` action to start processing, use the `ConfigName` action parameter to specify the name of the configuration file to use.
- Base-64 encode the configuration and send it with the `process` action when you start processing. Use the `Config` action parameter to include the base-64 encoded data.

A configuration that you specify in the `process` action overrides the `[Ingest]`, `[Analysis]`, `[Encoding]`, `[Transform]`, `[EventProcessing]`, and `[Output]` sections of the Media Server configuration file.

Tasks

To create a configuration for processing media, you must define *tasks*, which are individual operations that Media Server can perform.

- **Ingest tasks** bring media into Media Server so that it can be processed. Ingestion splits the data contained in a file or stream, for example video is split into images (video frames), audio, and

metadata. A configuration must contain exactly one ingest task.

- **Encoding tasks** make a copy of ingested media, or encode it into different formats. A configuration can contain any number of encoding tasks.
- **Analysis tasks** run analysis on ingested media, and produce metadata that describes the content of the media. A configuration can contain any number of analysis tasks.
- **Event stream processing (ESP) tasks** introduce additional custom logic into media analysis. For example, you can filter or deduplicate the information produced during analysis. A configuration can contain any number of ESP tasks.
- **Transform tasks** transform the data produced by other tasks, but do not modify its schema. A configuration can contain any number of transformation tasks.
- **Output tasks** send the data produced by Media Server to other systems. A configuration can contain any number of output tasks.

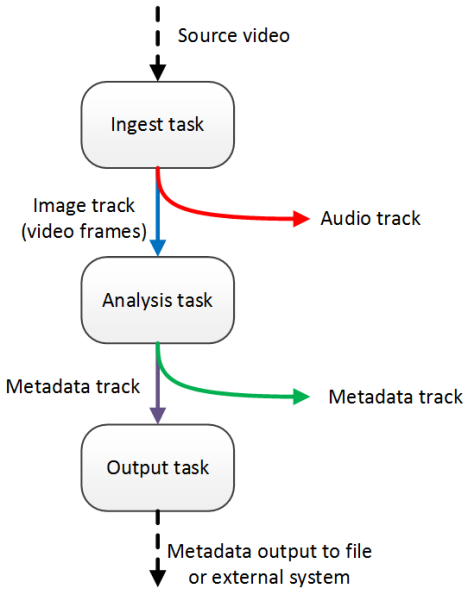
In a complete configuration, tasks that you define are connected. The output of an ingest task becomes the input for an analysis or encoding task. The output of an analysis task might become the input for an encoding, ESP, or output task. You can connect tasks in many different ways, which allows Media Server to be used to meet many different use cases.

Tracks

Information is passed between tasks in the form of *tracks*.

Tasks usually produce multiple tracks. For example, an ingest task can produce an image track that contains frames from the ingested video, and an audio track that contains the audio packages.

In the following example, an ingest task takes the source video and produces an image track and an audio track. The image track is used as the input for an analysis task. This could be object recognition, OCR, or another type of analysis that operates on images. In this example the audio track is not used. The analysis task produces some metadata tracks which contain information about the video content. One of the tracks is used as the input for an output task.



When you configure a task, you might need to specify the track(s) to use as the input for the task:

Task type	Default input tracks
Ingest	Ingest tasks do not accept input tracks. Specify the source media in the `process` action, when you start processing.
Encoding	Encoding tasks automatically use the first image and audio tracks produced by your ingest task, so you only need to specify the input for an encoding task if you want to encode different data, for example: • Some video sources contain more than one audio stream, to supply audio in multiple languages. You might want to encode an audio stream other than the default. • You might want to encode data produced by another task. For example, you might want to encode the keyframes identified during keyframe analysis, instead of all ingested frames.
Analysis	Most analysis tasks automatically analyze the first image or audio track produced by your ingest task, so in most cases you do not need to specify an input track. However, some analysis operations require additional data. For example, face recognition requires the metadata produced by face detection, so when you configure face recognition you must specify an input track.
ESP	You must always specify the input track(s) to use.
Transform	You must always specify the input track to use.
Output	Output tasks automatically use the default output tracks produced by your analysis tasks, but you can specify the input track(s) so that the output tasks use different tracks.

Records

Tracks contain *records*. A record is a collection of metadata that contains information about the media content. For example, face recognition produces records for each recognized face. All records in a track contain the same type of information and have the same schema.

When you process images or office documents, records contain a page number that indicates which page of the image or document the record is related to:

```
<record>
   <pageNumber>1</pageNumber>
   <trackname>OCR.Result</trackname>
   <OCRResult>
      <id>bcba9983-8dee-450e-9d67-3234e1a0c17a</id>
      <text>FOCUS ON ECONOMY</text>
      <region>
         <left>114</left>
         <top>476</top>
         <width>194</width>
         <height>19</height>
```

```
      </region>
      <confidence>84</confidence>
      <angle>0</angle>
      <source>image</source>
   </OCRResult>
</record>
```

When you process video, a record can represent data extracted from a single frame, or span multiple frames. Records created during video processing contain a timestamp that indicates which part of the video is described by the record. The timestamp includes a start time, peak time, duration, and end time. All of these values are provided in both epoch microseconds and ISO 8601 time format.

The peak time describes the time at which the event was most clearly visible in the video. For example, if you are running face recognition a face might appear in the video, remain in the scene for several seconds, and then leave the scene. The peak time describes the time at which the face was most clearly visible and at which face recognition recorded the highest confidence score.

The following is an example record produced by OCR, and shows an example timestamp:

```
<record>
   <timestamp>
      <startTime iso8601="2015-09-22T14:30:35.531005Z">1442932235531005</startTime>
      <duration iso8601="PT00H01M16.820000S">76820000</duration>
      <peakTime iso8601="2015-09-22T14:30:35.531005Z">1442932235531005</peakTime>
      <endTime iso8601="2015-09-22T14:31:52.351005Z">1442932312351005</endTime>
   </timestamp>
   <trackname>OCR.Result</trackname>
   <OCRResult>
      <id>bcba9983-8dee-450e-9d67-3234e1a0c17a</id>
      <text>FOCUS ON ECONOMY</text>
      <region>
         <left>114</left>
         <top>476</top>
         <width>194</width>
         <height>19</height>
      </region>
      <confidence>84</confidence>
      <angle>0</angle>
      <source>image</source>
   </OCRResult>
</record>
```

Analysis Task Output Tracks

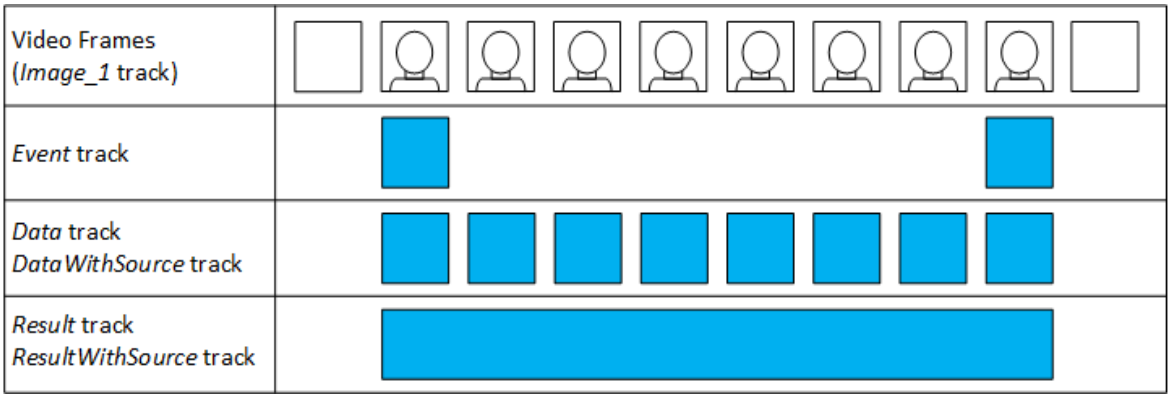
The events that occur in video usually span multiple frames. For example, a person, object, or logo might appear on screen and remain there for seconds or minutes. Media Server analysis tasks run analysis on single frames, but also follow events across multiple frames.

Analysis tasks can produce the following tracks:

- *Data* tracks contain a record for every frame of an event. For example, the data track from a face detection task contains one or more records for every frame in which a face is detected (Media Server creates multiple records when a frame contains more than one face). If a person remains in the scene for several seconds, this track could contain hundreds of records that represent the same face.
- *DataWithSource* tracks are similar to data tracks because they contain one or more records for every frame of an event. However, each record also includes the image analyzed by the task to produce the record.
- *Result* tracks contain a single record for each event. The result track summarizes the results of analysis for each event. For example, the result track from a face detection task contains a single record for each detected face. Each record has a start time, a peak time, a duration, and an end time, and can span many frames. Result tracks are, by default, used as input tracks for output tasks that you configure.
- *ResultWithSource* tracks are similar to result tracks because they contain a single record for each event. However, each record also includes the source image that produced the best result. For example, when you run face recognition the frame with the highest confidence score is added to the record. This frame corresponds to the peak time in the record timestamp.
- *Event* tracks contain records that describe the beginning or end of events in the video. For example, the event track for a face detection task contains a record when a face appears in the video, and another record when the face disappears from the scene. Event tracks are, by default, used as input tracks for output tasks that you configure.

The following diagram shows how Media Server creates records (represented by blue squares) when a person appears in a video.

- The Face Detection task creates records in the `Event` track when the person appears and disappears from the scene.
- The task creates one record in the `Data` and `DataWithSource` tracks for each analyzed frame. If there are multiple people in the scene, Media Server creates multiple records in the `Data` and `DataWithSource` tracks for each frame.
- The task creates a single record in the `Result` and `ResultWithSource` tracks for each event (in this case a detected face). This record spans the event and summarizes the analysis results.



For information about the tracks that are produced by Media Server tasks, and the data contained in each track, refer to the *Media Server Reference*.

Create a Configuration

To create a configuration, you need to consider:

- the media source, because this determines how to ingest the video.
- which analysis tasks you want to run, for example face recognition or speech-to-text.
- the way in which data will flow between tasks. For example, you cannot run face recognition without first running face detection. You must configure the input of the face recognition task to be the output from the face detection task.
- how to output data.

A Media Server configuration can include `[Ingest]`, `[Analysis]`, `[Transform]`, `[Encoding]`, `[EventProcessing]`, and `[Output]` sections. Each section contains a list of tasks that you want to run:

```
[Ingest]
IngestEngine=LibAvIngest

[Analysis]
AnalysisEngine0=FaceDetect
AnalysisEngine1=FaceRecognize
...

[Transform]
TransformEngine0=ResizeFaceImagesForEncoding
...

[EventProcessing]
EventProcessingEngine0=Deduplicate
...

[Encoding]
EncodingEngine0=Image
...

[Output]
OutputEngine0=IdolServer
...
```

These tasks are configured in other sections of the configuration file. The previous example defines an ingest task named `LibAvIngest` which you would configure in a section named `[LibAvIngest]`.

Every task configuration section contains the `Type` parameter to specify the engine to use to complete the task, and contains any settings that the engine requires to complete the task. For example:

```
[LibAvIngest]
Type=libav
```

Ingestion

Ingestion brings media into Media Server so that it can be processed. For example, if you ingest video then Media Server must extract the video and audio from the container and decode the streams so that they can be analyzed and transcoded.

Your configuration must include exactly one ingest task. For example:

```
[Ingest]
IngestEngine=LibAvIngest

[LibAvIngest]
Type=libav
```

This example specifies a single ingest task named `LibAvIngest`. The engine used to complete the task is the `libav` ingest engine. Notice that no source file or stream is specified in the configuration. You provide the path of a file or the URL of a stream to Media Server in the `Process` action when you start processing.

Ingest engines produce one or more image tracks and possibly audio tracks:

- Each image track is named `Image_`*n*, where *n* is a unique number. Tracks are numbered from 1.
- Each audio track is named `Audio_`*lang*`_`*n*, where *lang* is the language and *n* is a unique number. Tracks are numbered from 1. If the language is unknown, each track is named `Audio__`*n* (note the double underscore), where *n* is a unique number.

For example, if you ingest video from a TV broadcast, Media Server might produce one image track, `Image_1`, and three audio tracks: `Audio_French_1`, `Audio_English_2`, and `Audio_German_3`.

Analysis

The `[Analysis]` section of the configuration lists the analysis tasks that you want to run. A configuration can contain any number of analysis tasks. For example, you can run face detection and object recognition at the same time.

The following example includes a single analysis task named `OCRtext`. The task uses the `OCR` analysis engine:

```
[Analysis]
AnalysisEngine0=OCRtext

[OCRtext]
Type=ocr
Input=Image_1
```

An analysis engine accepts input of a particular type. For example, the OCR engine requires an image track, and the SpeakerID engine requires an audio track. The analysis engine only processes records from the track specified by the `Input` configuration parameter. In this example, the OCR engine processes the `Image_1` track produced by the ingest engine. If you do not specify an input track, the engine uses the first track of the required type produced by the ingest engine. The engine would automatically attempt to process the `Image_1` track.

To find out what type of input is required by an analysis engine, send `action=getstatus` to Media Server. The response includes information about the input and output tracks for the analysis engines.

The input could be an output track from another engine, provided the track type is compatible. In the following example, the input of the face recognition task is the `ResultWithSource` output track of the face detection task:

```
[Analysis]
AnalysisEngine0=FaceDetect
AnalysisEngine1=FaceRecognize

[FaceDetect]
Type=FaceDetect
...

[FaceRecognize]
Type=FaceRecognize
Input=FaceDetect.ResultWithSource
...
```

To be used as the input for a task, a track must provide at least the required record types. For information about the output tracks produced by Media Server engines, use `action=getstatus` or refer to the *Media Server Reference*.

Transform

The `[Transform]` section of the configuration lists the transformation tasks that you want to run. A configuration can contain any number of transformation tasks.

A transformation task requires a single input, transforms the data in some way, and produces a single output with the same schema as the input. For example, you can use a transformation task to resize keyframes extracted by keyframe analysis, before sending them to the image encoder and writing them to disk.

The following example includes a single transformation task named `ScaleKeyframes`. The task uses the `Scale` transformation engine:

```
[Transform]
TransformEngine0=ScaleKeyframes

[ScaleKeyframes]
Type=Scale
Input=Keyframe.ResultWithSource
ImageSize=300,0
```

All transformation engines produce a single output track with the name *`TaskName`*`.Output`, where *`TaskName`* is the name of the transformation task.

Encoding

The `[Encoding]` section lists the names of tasks that produce encoded copies of the media processed during the session. A configuration can contain any number of encoding tasks. For example:

```
[Encoding]
EncodingEngine0=MyRollingBuffer

[MyRollingBuffer]
Type=rollingbuffer
// rolling buffer configuration
```

This example specifies a single encoding task named `MyRollingBuffer`.

An encoding engine accepts image and/or audio tracks produced by an ingest or analysis engine. For example, you can:

- make a copy of ingested video.
- make a copy of ingested video at a different resolution or bitrate to the source.
- encode the output of an analysis engine - for example use the Image Encoder to write the keyframes identified by keyframe analysis to disk.

All encoding tasks produce a single output track with the name *`TaskName`*`.Proxy`, where *`TaskName`* is the name of the encoding task. This track contains information about the encoded media. You can output this information alongside your analysis results so that a front-end application can open and display the encoded media that shows a specific event, such as an identified news story or a recognized face.

Output

The `[Output]` section lists the tasks that output information produced by Media Server. Usually a configuration contains at least one output task. For example:

```
[Output]
OutputEngine0=IDOL
OutputEngine1=Vertica

[IDOL]
Type=idol
// idol engine options

[Vertica]
...
```

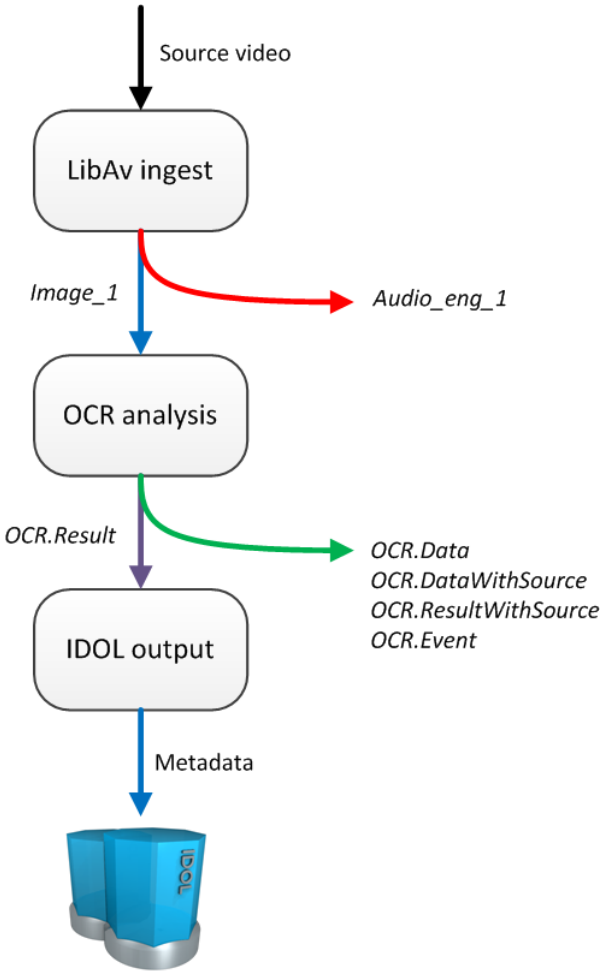
This example specifies two output tasks, named `IDOL` and `Vertica`.

All output tasks can include the configuration parameter `Input`. This specifies a comma-separated list of tracks that you want to output to an external system. If you do not specify a list of tracks, Media Server outputs a default selection of tracks. For information about whether a track is output by default, refer to the *Media Server Reference*.

Output engines do not produce records, and therefore do not have output tracks.

Example Configuration

The following diagram shows a simple configuration for performing OCR on a video stream and sending the results to IDOL Server. The setup includes an ingest engine, an analysis engine, and an output engine. These are usually the minimum required for any configuration.



1. Media Server receives a `Process` action, which starts a new session. The Libav ingest engine receives video from the source specified in the action. The Libav engine produces an image track and audio track.
2. The image track is used as the input for the OCR analysis engine.
3. The audio track cannot be used by the OCR analysis engine so is discarded.
4. The OCR engine produces several output tracks. The Result track contains the OCR results and is used as the input for the IDOL output task.
5. The other tracks produced by the OCR engine are not required and so are discarded.
6. The IDOL output engine indexes the data produced by Media Server into IDOL.

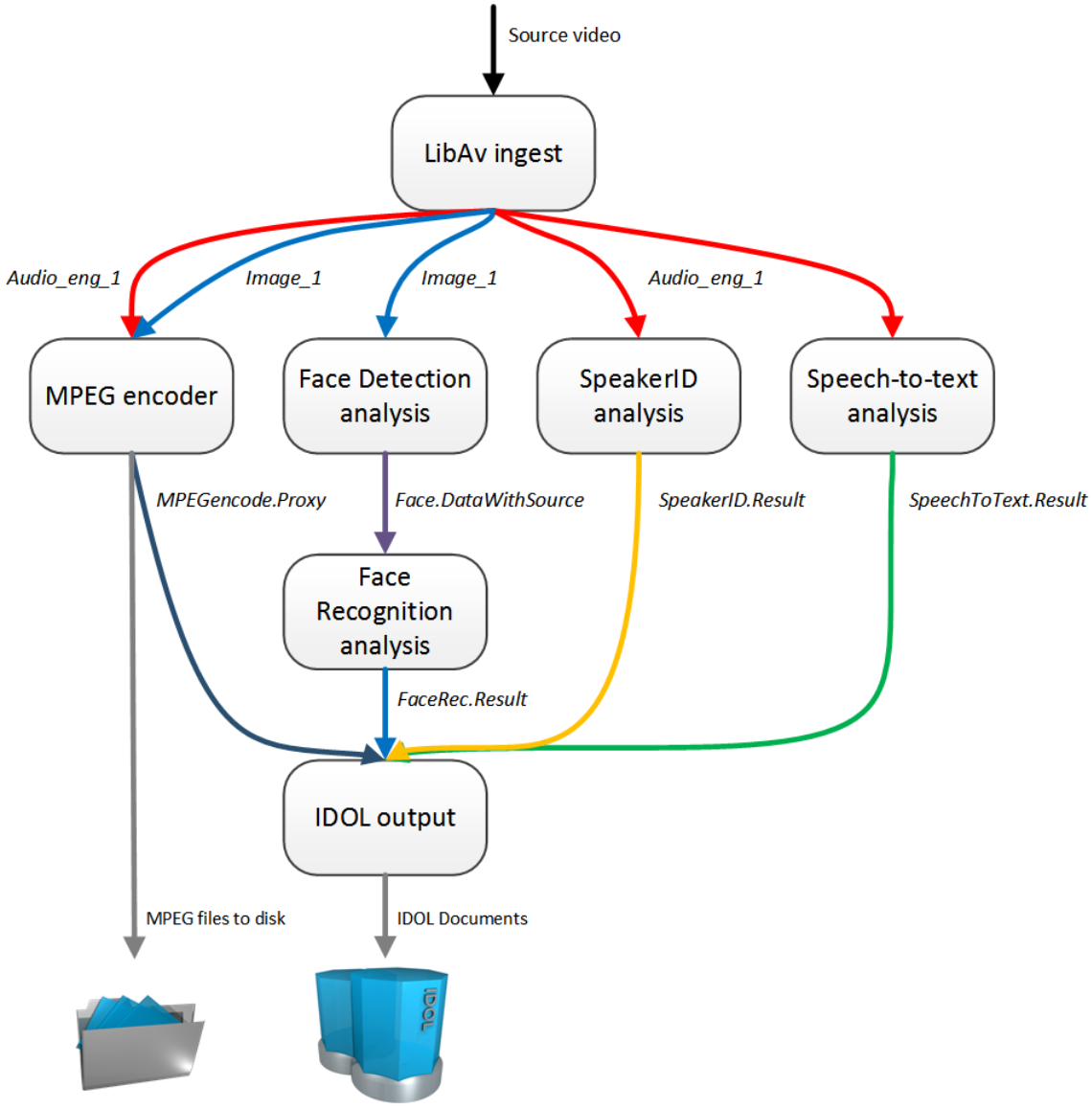
A Media Server configuration that matches this process is shown below.

- The `OCRtext` task includes the parameter **`Input=Image_1`**, which specifies that the input for the OCR task is the `Image_1` track from the ingest engine. Setting the `Input` parameter is optional, because by default Media Server processes the first available track of the correct type.
- The `IDOLOutput` task includes the parameter **`Input=OCRtext.Result`**. This specifies that the input for the IDOL output task is the `Result` track from the `OCRtext` task.

```
[Ingest]
IngestEngine=LibAvIngest

[LibAvIngest]
Type=libav

[Analysis]
AnalysisEngine0=OCRtext

[OCRtext]
Type=ocr
Input=Image_1

[Output]
OutputEngine0=IDOLOutput

[IDOLOutput]
Input=OCRtext.Result
Type=idol
Mode=time
OutputInterval=60
XslTemplate=./xsl/toIDX.xsl
```

Example Configuration - Advanced

The following diagram shows a more complex configuration, which includes an encoding task and multiple analysis tasks.



1. Media Server receives a `Process` action, which starts a new session. The Libav ingest engine receives video from the source specified in the action. The Libav engine produces an image track and an audio track.
2. The image track is used as the input for the MPEG encoder and Face Detection analysis tasks.
3. The audio track is used as the input for the MPEG encoder, and the Speaker ID and Speech-to-text analysis tasks.

4. The MPEG encoder writes a copy of the video to disk as a set of MPEG files. It also produces proxy information that is available to the output engine.
5. The face detection engine produces a track that contains information about detected faces. This is used as the input for the face recognition analysis engine.
6. The output tracks from the analysis engines are used as the input for the IDOL Server output engine. The analysis engines all produce multiple tracks, some of which are not used.
7. The IDOL output engine transforms the records produced by the MPEG encoder and analysis engines into documents and indexes the information into IDOL Server.

A Media Server configuration that matches this process is shown below.

```
[Ingest]
IngestEngine=LibAvIngest

[LibAvIngest]
Type=libav


[Encoding]
EncodingEngine0=MPEGencode

[MPEGencode]
Type=mpeg
OutputPath=\\server\folder\file.mpg
URLBase=https://www.myserver.com/folder/
Segment=TRUE


[Analysis]
AnalysisEngine0=Face
AnalysisEngine1=FaceRec
AnalysisEngine2=SpeakerID
AnalysisEngine3=SpeechToText

[Face]
Type=FaceDetect

[FaceRec]
Type=FaceRecognize
Input=Face.DataWithSource

[SpeakerID]
Type=SpeakerID
SpeakerIDServers=10.0.0.2:13000
TemplateSet=speakers.ats

[SpeechToText]
Type=SpeechToText
Language=ENUK
```

```
Mode=relative
ModeValue=0.90
SpeechToTextServers=10.0.0.2:13000


[Output]
OutputEngine0=IDOLOutput

[IDOLOutput]
Type=IDOL
Input=FaceRec.Result,SpeakerID.Result,SpeechToText.Result,MPEGencode.Proxy
Mode=time
OutputInterval=60
XslTemplate=./xsl/toIDX.xsl
SavePostXML=true
XMLOutputPath=./output/idol/
```

Image and Video Processing

This section shows which Media Server features are supported for different types of media.

- **Image** includes single image files and images that are extracted from documents such as PDF files.
- **Video** includes video files and video streams.

Analysis	Image	Video
Face detection, recognition, and analysis	✓	✓
Optical Character Recognition	✓	✓
Image classification	✓	✓
Object detection	✓	✓
Object recognition	✓	✓
Speech analysis (speech-to-text and speaker ID)	✗	✓
News segmentation	✗	✓
Vehicle analysis (ANPR and vehicle model detection)	✓	✓
Scene analysis	✗	✓

Keyframe extraction	✗	✓
Barcode detection and recognition	✓	✓
Color analysis	✓	✓
Image hash generation	✓	✓
Encoding	**Image**	**Video**
MPEG encoder (to file or UDP stream)	✗	✓
Image encoder	✓	✓
Rolling buffer encoder	✗	✓
Event Stream Processing	**Image**	**Video**
Filter	✓	✓
Combine	✓	✓
Deduplicate	✓	✓
And	✓	✓
AndThen	✗	✓
AndNot	✓	✓
AndNotThen	✗	✓
Or	✓	✓
Output	**Image**	**Video**
ACI Response	✓	✓
Files on disk	✓	✓
Connector Framework Server	✓	✓

IDOL Server	✓	✓
Vertica	✓	✓
ODBC database	✓	✓
HTTP POST	✓	✓
Broadcast Monitoring	✗	✓
Milestone XProtect	✗	✓

Start Processing

To start processing, send the `process` action to Media Server.

Media Server adds your request to the `process` action queue. When the request reaches the top of the queue and a thread is available, Media Server starts to ingest the media and perform the configured tasks.

To start processing

- Send the `process` action to Media Server. Set the following parameters:

  `Source` (Set either `Source` or `SourceData`)

  The media source to process.

  - To ingest a file, specify path to the file.
  - To ingest a video stream, specify a URL.
  - To ingest video from a DirectShow source, such as a video capture card, specify the name of the video device.
  - To ingest video from Milestone XProtect, specify the camera GUID.

  `SourceData` (Set either `Source` or `SourceData`)

  The media file to process (as binary data). For information about sending data to Media Server, see "Send Actions to Media Server" on page 66.

  `Persist` Specifies whether the action restarts in the event that processing stops for any reason. For example, if you are processing video from an RTSP camera which becomes unreachable and `persist=true`, Media Server will wait for the stream to become available again. Persistent actions only stop when you stop them using the `QueueInfo` action with `QueueAction=stop`, or when Media Server finishes processing the media.

  `Config` (Optional) A base-64 encoded configuration that defines the tasks to run. This overrides the Media Server configuration file.

  `ConfigName` (Optional) The name of a configuration file that defines the tasks to run. This overrides the Media Server configuration file. The file must be stored in the directory specified by the `ConfigDirectory` parameter in the `[Paths]` section of the configuration file.

  For example,

```
http://localhost:14000/action=Process&Source=.\video\broadcast.mpeg
                                      &ConfigName=broadcast
```

  This action is asynchronous, so Media Server returns a token for the request. You can use the `QueueInfo` action, with `QueueName=Process` to retrieve more information about the request.

Verify Media Server is Processing

You can run the action `GetLatestRecord` to verify that Media Server is processing media. This action returns the latest records that have been added to the tracks you specify in the action.

For example, send the following action:

```
http://localhost:14000/action=GetLatestRecord
                                  &Token=...
                                  &Tracks=Keyframe.Result,OCR.Result
```

- where the `token` parameter specifies the asynchronous action token returned by the `process` action,
- and `tracks` specifies the tracks to retrieve the latest records from. You can set this parameter to an asterisk (`*`) to retrieve the latest records from all tracks.

Stop Processing

When you process a file, the `process` action finishes when Media Server reaches the end of the file. However, you might be processing a video stream that does not end. In this case, you might want to stop processing.

To stop processing video

- Use the `QueueInfo` action, with the following parameters:

`QueueName`	The name of the queue. Processing is initiated using the `process` action, so set `QueueAction=process`.
`QueueAction`	The action to perform on the queue. To stop processing, set `QueueAction=stop`.
`Token`	(Optional) The token for the request that you want to stop. If you don't specify a token, Media Server stops the task that is currently processing.

For example,

```
http://localhost:14000/action=queueinfo
                        &queuename=process
                        &queueaction=stop
                        &token=MTAuMi4xMDQuODI6MTQwMDA6UFJPQ0VTUzoxMTgzMzYzMjQz
```

Media Server returns a response stating whether the request was successfully completed.

Optimize Analysis Performance with Parallel Processing

Media Server can use multiple threads to perform some analysis tasks on video. (If your source media is a single image or document, Media Server always uses one thread for each analysis task).

Using multiple threads allows Media Server to process multiple video frames simultaneously. You might want to use multiple threads for the following reasons:

- Some analysis operations are resource-intensive and cannot process video frames as quickly as they are ingested. To process a video in real time, Media Server might need to drop frames. Allowing Media Server to use more than one thread means that it can process a greater proportion of the frames and this can increase accuracy.
- If you have configured Media Server to process a video file without dropping frames, allowing Media Server to use more threads means that the media is processed in less time.

Media Server can use multiple threads for performing:

- face detection.
- optical character recognition.
- object recognition.
- number plate recognition.
- image classification when you use `Bayesian` or `MaxVote` classifiers, but not `CNN` classifiers.

To specify the number of threads that an analysis task can use, set the `NumParallel` configuration parameter in the task section of your configuration. For example:

```
[FaceDetect]
Type=FaceDetect
Database=Faces
NumParallel=2

[NumberPlate]
Type=NumberPlate
Location=UK
NumParallel=2
```

It is important to consider the total number of threads that Media Server will use. For example, a configuration for monitoring a television broadcast might include the following tasks:

- Face detection, with `NumParallel=2` (so face detection will use two threads).
- Face recognition to recognize known faces (one thread).
- Object recognition to detect corporate logos, with `NumParallel=2` (two threads).
- Speech-to-text (one thread)
- Optical character recognition with `NumParallel=1` (one thread).
- An encoding task to write the video to disk or a rolling buffer (one thread).

This demonstrates that a single `process` action can easily consume many threads (at least eight in the previous example). If you have configured Media Server to run multiple process actions simultaneously, then the total number of threads consumed is multiplied by the number of actions that you are running. It is important to install Media Server on a server with sufficient resources.

The optimum number of threads to use for processing depends on the number of CPU cores your server has available. Increasing the number of threads used by Media Server up to this limit will increase throughput. For example if your server has 20 CPU cores then allowing Media Server to consume 16 threads rather than 8 should almost double throughput. Using more threads than the server has CPU cores may increase throughput but the improvement will be smaller for each additional thread. Using significantly more threads than the server has CPU cores is not recommended and may decrease performance.

Part II: Ingest Media

Before you can start processing media, you must create a configuration that defines which operations to perform. This section describes how to create the `[Ingest]` section of your configuration, which configures Media Server to ingest media.

- "Video Files and Streams"
- "Images and Documents"
- "Cameras and Third-Party Systems"

Chapter 7: Video Files and Streams

This section describes how to ingest video files and streams. When you ingest video, an ingest engine receives video input and produces demuxed and decoded streams ready for processing by other engines.

Supported Codecs and Formats

This page lists the audio and video codecs, and the file formats, supported by Media Server. To ingest files that use these codecs and formats, use a LibAv ingest task (see "Ingest Video from a File" on page 102).

Other codecs and file formats might work, but they are not supported.

Video Codecs

- MPEG1
- MPEG2
- MPEG4 part 2
- MPEG4 part 10 (Advanced Video Coding) (h264)
- MPEGH part 2 (High Efficiency Video Coding) (h265)
- Windows media 7
- Windows media 8

Audio Codecs

- MPEG audio layer 1
- MPEG audio layer 2
- MPEG audio layer 3
- MPEG 4 audio (Advanced Audio Coding) (AAC)
- PCM (wav)
- Windows media audio 1
- Windows media audio 2
- AC3 (ATSC A/52)

File Formats

- MPEG packet stream (for example `.mpg`)
- MPEG2 transport stream (for example `.ts`)
- MPEG4 (for example `.mp4`)
- WAVE (Waveform Audio) (`.wav`)
- asf (Windows media) (`.asf`, `.wmv`)
- raw aac (`.aac`)
- raw ac3 (`.ac3`)

Choose the Rate of Ingestion

Some analysis operations are resource-intensive and cannot process video frames as quickly as they can be ingested. For this reason, and because processing every frame does not necessarily improve detection performance or capture rates, analysis tasks usually do not process every frame of a video.

The interval at which an analysis task selects frames to be analyzed is called the *sample interval*. For example, if a task has a sample interval of 125 milliseconds then any two frames analyzed by the task will be separated by at least 125 milliseconds. This means that at most eight frames are analyzed for every second of video.

Media Server uses default sample intervals that have been chosen to produce good accuracy in most cases. HPE recommends using the default values but you can increase or decrease the sample interval for an analysis task by setting the `SampleInterval` configuration parameter.

If you are processing a live stream, Media Server is ingesting video at a fixed rate and must keep up with the video. This means that if your Media Server does not have sufficient resources, it will not be able to process all of the frames that have been selected for analysis. In this case, your analysis tasks will drop some of the frames that were selected to ensure that analysis keeps up with the video. This might cause a reduction in accuracy. When you are processing a live stream, you cannot change the rate at which video is ingested, so to prevent frames being dropped the only solution is to increase the resources (CPU cores) available to Media Server.

If you are processing video that is ingested from a file, you can choose the rate at which video is ingested. The rate at which Media Server ingests video is controlled by the `IngestRate` configuration parameter. You can set this parameter to one of the following values:

- **`1`** - This is the default value and must be used if you are processing a live stream. Media Server ingests video at normal (playback) speed. Assuming there are no external limitations such as network bandwidth, Media Server ingests one minute of video every minute. If an analysis task cannot process frames at this speed, it will drop frames.
- **`0`** - The rate at which video is ingested is determined by the slowest analysis task that you have configured. Analysis tasks will not drop frames that have been selected for analysis. Some analysis tasks are very fast, but some are slower and can analyze only one or two frames per second per thread. With `IngestRate=0`, the amount of time required to ingest a video file depends on the analysis tasks that you have configured and the resources available to Media Server.

Tip: The sample interval and rate of ingestion have no effect on the speed at which a video frame is analyzed. The speed of analysis is determined by the resources available to Media Server. If your server has sufficient resources (CPU cores) and an analysis task supports it, you can set the `NumParallel` parameter. This specifies the number of video frames for an analysis task to process concurrently. Processing frames in parallel can increase the total number of frames processed in a certain time. Depending on the ingest rate, this will either result in fewer frames being dropped or result in the file being processed in less time.

The following table summarizes the combinations that are available for `IngestRate` and `SampleInterval`:

Options	Default SampleInterval	SampleInterval = 0
IngestRate=1	HPE recommends using `IngestRate=1` and the default sample intervals for processing live streams. Media Server does not attempt to process every frame, but selects frames for analysis based on the specified sample interval. The default sample interval should provide good accuracy in most cases. However, Media Server might drop sampled frames to ensure that analysis occurs at normal (playback) speed. If your Media Server does not have sufficient resources, it will not be able to analyze all of the frames that are sampled and will drop frames, causing a reduction in accuracy.	Media Server attempts to analyze every frame but frames are dropped if the analysis tasks you have configured cannot process the frames at normal (playback) speed. HPE does not recommend the combination of `IngestRate=1` and `SampleInterval=0` in any scenario because Media Server is likely to drop large numbers of frames.
IngestRate=0	HPE recommends `IngestRate=0` and the default sample intervals for processing video from files. Media Server does not attempt to process every frame, but selects frames for analysis based on the specified sample interval. The default sample interval should provide good accuracy in most cases. If analysis is slow, Media Server ingests the file at a slower rate and does not drop frames. If your Media Server does not have sufficient resources, it will take longer to process the file, but accuracy will be maintained.	Ensures that all frames are analyzed and no frames are dropped. HPE does not recommend the combination of `IngestRate=0` and `SampleInterval=0` because in most cases this will take much longer and may not result in a large increase in accuracy. Use this mode only if it is critical that every frame is analyzed.

Ingest Video from a File

To ingest video from a file, follow these steps.

To ingest video from a file

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Ingest]` section, set the following configuration parameters:

`IngestRate`	The rate at which Media Server ingests video. If you are ingesting video from files, HPE recommends setting this parameter to **`0`**. For more information about choosing an ingest rate, see "Choose the Rate of Ingestion" on page 100.
`IngestEngine`	The name of a section in the configuration file that will contain the ingestion settings

   For example:

   ```
   [Ingest]
   IngestRate=0
   IngestEngine=VideoFile
   ```

3. Create a new section in the configuration file to contain the ingestion settings (using the name you specified with the `IngestEngine` parameter). Then, set the following parameters:

`Type`	The ingest engine to use. Set this parameter to **`libav`**.
`IngestTime`	(Optional) The ingest engine produces output tracks that contain timestamped records. You can use this parameter to control the timestamp values, which is useful if you are ingesting video that was originally broadcast at an earlier time. Specify the time in epoch seconds.
`StartOffset`	(Optional) The position in the video at which to start processing. For example, to ignore the first minute of a video file, set this parameter to **`60seconds`**.
`MaximumDuration`	(Optional) The maximum amount of video and audio to ingest, before stopping ingestion. If you do not set this parameter, there is no limit on the amount of video and audio that is ingested. You can set this parameter to ingest part of a video file. If the maximum duration is reached, Media Server stops ingestion but does finish processing any frames that have already been ingested.

   For example:

   ```
   [VideoFile]
   Type=libav
   ```

   Tip: There is no configuration parameter to specify the source. You must specify the path of

the source file or the IP address of the stream that you want to ingest as the value of the `Source` parameter in the `Process` action, when you start processing.

For more information about the parameters you can set to configure ingestion, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Ingest Video from a Stream

To ingest video from a stream, follow these steps.

To ingest video from a stream

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Ingest]` section, set the following configuration parameters:

`IngestRate`	The rate at which Media Server ingests video. If you are ingesting video from a stream, you must set this parameter to **`1`**.
`IngestEngine`	The name of a section in the configuration file that will contain the ingestion settings

For example:

```
[Ingest]
IngestRate=1
IngestEngine=VideoStream
```

3. Create a new section in the configuration file to contain the ingestion settings, and set the following parameters:

`Type`	The ingest engine to use. Set this parameter to **`libav`**.
`IngestTime`	(Optional) The ingest engine produces output tracks that contain timestamped records. You can use this parameter to control the timestamp values, which is useful if you are ingesting video that was originally broadcast at an earlier time. Specify the time in epoch seconds.
`MaximumDuration`	(Optional) The maximum amount of video and audio to ingest, before stopping ingestion. If you do not set this parameter, there is no limit on the amount of video and audio that is ingested. If the maximum duration is reached, Media Server stops ingestion but does finish processing any frames that have already been ingested.

For example:

```
[VideoStream]
Type=libav
```

> **Tip:** There is no configuration parameter to specify the source. You must specify the IP address of the stream that you want to ingest as the value of the `Source` parameter in the `Process` action, when you start processing.

For more information about the parameters you can set to configure ingestion, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 8: Images and Documents

This section describes how to ingest image files and documents, such as Microsoft Word or Adobe PDF files.

Introduction

Media Server uses IDOL KeyView to process images and documents. Media Server can ingest most image files, including multi-page images, and also accepts documents such as PDF files and PowerPoint presentations.

There are some differences in the way that these files are ingested:

- When you ingest an image the ingest engine creates a single record that contains the image.
- When you ingest a multi-page image the ingest engine creates a record for each page, and each record contains the image from the relevant page.
- When you ingest a presentation (.PPT, .PPTX, or .ODP) file, the ingest engine creates a record for each slide contained within the presentation. Each record contains an image of the slide.
- When you ingest a PDF file, the ingest engine creates a record for each page of the document. Each record contains an image of the full page. Each record also contains information about any text elements that are associated with the page (PDF files can include embedded text, which is stored as encoded data, such as UTF8, instead of as an image of the text). For each text element, Media Server extracts the text, its position on the page, and its orientation.
- For other document formats, Media Server does not extract information about pages or the position of text elements. In some cases this is because the file formats do not contain this information. When you ingest a document in a format other than PDF, the ingest engine creates a record for each image that is embedded in the document. Each record contains the embedded image and any text that follows the image, as extracted by KeyView. In this case, Media Server does provide co-ordinates for text elements contained in the document, but this is only so that the information has a consistent form for all input file formats, and the co-ordinates do not represent the position of the text in the original document.

When you ingest images and documents, Media Server does not perform tracking between images. Images are considered to be independent and are not considered as a sequence.

For information about which analysis operations you can run on images and documents, see "Image and Video Processing" on page 91.

Supported File Formats

Media Server can ingest all standard image formats:

- TIFF
- JPEG
- PNG
- GIF (transparency information is not supported)
- BMP (compressed BMP files are not supported) and ICO
- PBM, PGM, and PPM

Additionally, Media Server uses HPE KeyView to support other document formats, including:

- Adobe PDF
- Microsoft Word Document (.DOC and .DOCX)
- Microsoft Excel Sheet (.XLS and .XLSX)
- Microsoft PowerPoint Presentation (.PPT and .PPTX)
- OpenDocument Text (.ODT)
- OpenDocument Spreadsheet (.ODS)
- OpenDocument Presentation (.ODP)
- Rich Text (RTF)

Ingest Images and Documents

To configure a task to ingest an image file or document

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Ingest]` section, set the `IngestEngine` parameter to the name of a section in the configuration file that will contain the ingestion settings. For example:

   ```
   [Ingest]
   IngestEngine=SingleImageOrDocument
   ```

3. Create a new section in the configuration file to contain the ingestion settings, and set the following parameters:

   `Type` The ingest engine to use. Set this parameter to **`image`**.

   For example:

   ```
   [SingleImageOrDocument]
   Type=image
   ```

> **Tip:** There is no configuration parameter to specify the source. You must specify the path of the source file that you want to ingest as the value of the `Source` parameter in the `Process` action, when you start processing.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Output Records

This section describes the records that are produced when you ingest an image or document file.

Images and Multi-Page Images

The following sample XML shows a record produced when you ingest an image, multi-page image such as a TIFF file, or a presentation file (.PPT, .PPTX, .ODP).

```
<record>
  <pageNumber>1</pageNumber>
  <trackname>Image_1</trackname>
  <Page>
    <image>
      <imagedata format="PNG">...</imagedata>
      <width>222</width>
      <height>140</height>
      <pixelAspectRatio>1:1</pixelAspectRatio>
    </image>
    <pagetext/>
  </Page>
</record>
```

The record contains the following information:

- The `pagenumber` element describes the page that the record is associated with. Most image files have a single page but formats such as TIFF support multiple pages.
- The `image` element contains the image data, and provides the `width` and `height` of the image. The `pixelAspectRatio` element describes the shape of the pixels that make up the image, for example 1:1 pixels are square.

Documents

If you ingest a document such as a PDF file, the output might also include the text extracted from text elements:

```
<record>
  <pageNumber>1</pageNumber>
  <trackname>Image_1</trackname>
  <Page>
    <image>
```

```
      <imagedata format="PNG">...</imagedata>
      <width>892</width>
      <height>1260</height>
      <pixelAspectRatio>1:1</pixelAspectRatio>
    </image>
    <pagetext>
      <element>
        <text>Some text</text>
        <region>
          <left>115</left>
          <top>503</top>
          <width>460</width>
          <height>41</height>
        </region>
        <angle>0</angle>
      </element>
      ...
    </pagetext>
  </Page>
</record>
```

The `pagetext` element contains information about associated text elements. If the ingested media was a PDF file, each record represents a page. If the ingested media was another type of document the record represents an embedded image and the text that follows it, up to the next embedded image.

Each `element` element describes a text element and contains the following data:

- The `text` element contains the text from the text element.
- The `region` element provides the position of the text element on the page.

  Note: The region information is accurate only if the ingested document was an Adobe PDF file.

- The `angle` element provides the orientation of the text.

Information about text elements is used by the OCR analysis engine, which automatically combines the text elements with the text extracted from images, to produce a complete transcript of the text that appears on the page.

Chapter 9: Cameras and Third-Party Systems

Media Server can ingest video from cameras and third-party video management systems.

Ingest Video from a DirectShow Device

Media Server can ingest video directly from a DirectShow device, such as a video capture card or camera.

Note: Ingestion from a DirectShow device is supported only on Windows.

To ingest video from a DirectShow device

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Ingest]` section, set the `IngestEngine` parameter to the name of a section in the configuration file that will contain the ingestion settings. For example:

   ```
   [Ingest]
   IngestEngine=CaptureCard
   ```

3. Create a new section in the configuration file to contain the ingestion settings, and set the following parameters:

`Type`	The ingest engine to use. Set this parameter to **`libav`**.
`Format`	The video format. Set this parameter to **`dshow`**.
`IngestTime`	(Optional) The ingest engine produces output tracks that contain timestamped records. You can use this parameter to control the timestamp values, which is useful if you are ingesting video that was originally broadcast at an earlier time. Specify the time in epoch seconds.

   For example:

   ```
   [CaptureCard]
   Type=libav
   Format=dshow
   ```

   Tip: There is no configuration parameter to specify the source. You must specify the name of the DirectShow device that you want to use as the value of the `Source` parameter in the `Process` action, when you start processing.

For more information about the parameters you can set to configure ingestion, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Ingest Video from Milestone XProtect

Media Server can ingest live video from Milestone XProtect Enterprise and Milestone XProtect Corporate.

Media Server connects to the Milestone XProtect recording server to obtain video. The recording server requires user authentication, so you must specify a user name and password in the Media Server configuration. HPE recommends that you encrypt passwords before entering them into the configuration file. For information on how to do this, see "Encrypt Passwords" on page 56.

To ingest video from Milestone XProtect

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Ingest]` section, create an ingest task by setting the `IngestEngine` parameter.

   ```
   [Ingest]
   IngestEngine=XProtect
   ```

3. Create a configuration section to contain the task settings. Type the task name inside square brackets, and then set the following parameters:

`Type`	The ingest engine to use. Set this parameter to **`milestone`**.
`RecorderHost`	The host name or IP address of the XProtect recording server.
`RecorderPort`	The port of the XProtect recording server.

   For example:

   ```
   [XProtect]
   Type=milestone
   RecorderHost=XProtect
   RecorderPort=7563
   ```

4. Configure how the ingest engine should authenticate with the Milestone XProtect system.

   Note: To ingest video from Milestone XProtect Corporate, you must use NTLM authentication against the Milestone XProtect authentication server. To ingest video from Milestone XProtect Enterprise, you can use any of the following options.

   - To use NTLM authentication against a Milestone authentication server, set the following parameters:

`SOAPAuthentication`	Set this parameter to **`true`**. The engine authenticates against the

	XProtect authentication server.
`NTLMUsername`	The NT user name to use to retrieve video.
`NTLMPassword`	The NT password to use to retrieve video.
`AuthenticationHost`	The host name or IP address of the XProtect authentication server.
`AuthenticationPort`	The port of the XProtect authentication server.

- To use Milestone credentials (Basic authentication) against a Milestone authentication server, set the following parameters:

`SOAPAuthentication`	Set this parameter to **`true`**. The engine authenticates against the XProtect authentication server.
`BasicUsername`	The user name to use to retrieve video.
`BasicPassword`	The password to use to retrieve video.
`AuthenticationHost`	The host name or IP address of the XProtect authentication server.
`AuthenticationPort`	The port of the XProtect authentication server.

- To use Milestone credentials (Basic authentication) against the Milestone recording server, set the following parameters:

`SOAPAuthentication`	Set this parameter to **`false`**. The engine sends credentials to the recording server.
`BasicUsername`	The user name to use to retrieve video.
`BasicPassword`	The password to use to retrieve video.

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

You can now start ingesting video. When you use the `process` action, you must set the `source` parameter to a Milestone camera GUID. For information about how to find the GUID, refer to the Milestone documentation. Unless you are using SOAP authentication, you can specify the camera ID instead. For more information about how to start processing, see "Start Processing" on page 93.

Part III: Analyze Media

This section describes how to configure the analysis operations that Media Server can perform.

Chapter 10: Detect, Recognize, and Analyze Faces

Media Server can run several types of face analysis including face detection and face recognition.

Introduction

Face Detection detects faces and returns their location.

Face Recognition identifies people by comparing detected faces to faces in your training database.

After detecting faces, Media Server can also:

- estimate demographic information such as age, gender, and ethnicity.
- report information such as the facial expression, whether the person's eyes are open or closed, and whether the person is wearing spectacles.
- locate the region containing a person's clothing. You can then use this information with other analysis tasks, for example you can run color analysis to identify the dominant colors of the clothing.

Note: Media Server can analyze faces for demographics and facial expression only when the person is looking directly at the camera.

You can run face detection and face analysis operations out-of-the-box. To run face recognition, you must train Media Server using images of people that you want to identify. For information about how to train Media Server, see "Train Media Server to Recognize Faces" on page 116.

Detect Faces

To detect faces that appear in media, follow these steps.

To detect faces in media

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=FaceDetect
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`FaceDetect`**.
`Input`	(Optional) The image track to process. If you do not set this parameter, Media Server processes the first track of the correct type.
`Region`	(Optional) The region that you want to search for faces.
`RegionUnit`	(Optional) The units to use for specifying the region (default `pixel`). To specify the position and size of the region as a percentage of the media size, set this parameter to **`percent`**.
`FaceDirection`	(Optional) Media Server can detect faces looking in any direction, but if you know that all faces will look directly at the camera you can set **`FaceDirection=Front`**. This is faster and might produce fewer false detections.
`ColorAnalysis`	(Optional) A Boolean value that specifies whether to perform color analysis on detected faces. This can reduce the number of false detections.
`MinSize`	(Optional) The minimum expected size of faces in the video (in pixels unless you set the `SizeUnit` parameter). Increasing the minimum size can increase processing speed.
`SizeUnit`	(Optional) The units to use for setting the `MinSize` parameter (default `pixel`). To specify the size as a percentage of the smallest image side, set this parameter to **`percent`**.

   For example:

   ```
   [FaceDetect]
   Type=FaceDetect
   FaceDirection=Front
   ColorAnalysis=TRUE
   ```

```
MinSize=200
SizeUnit=pixel
```

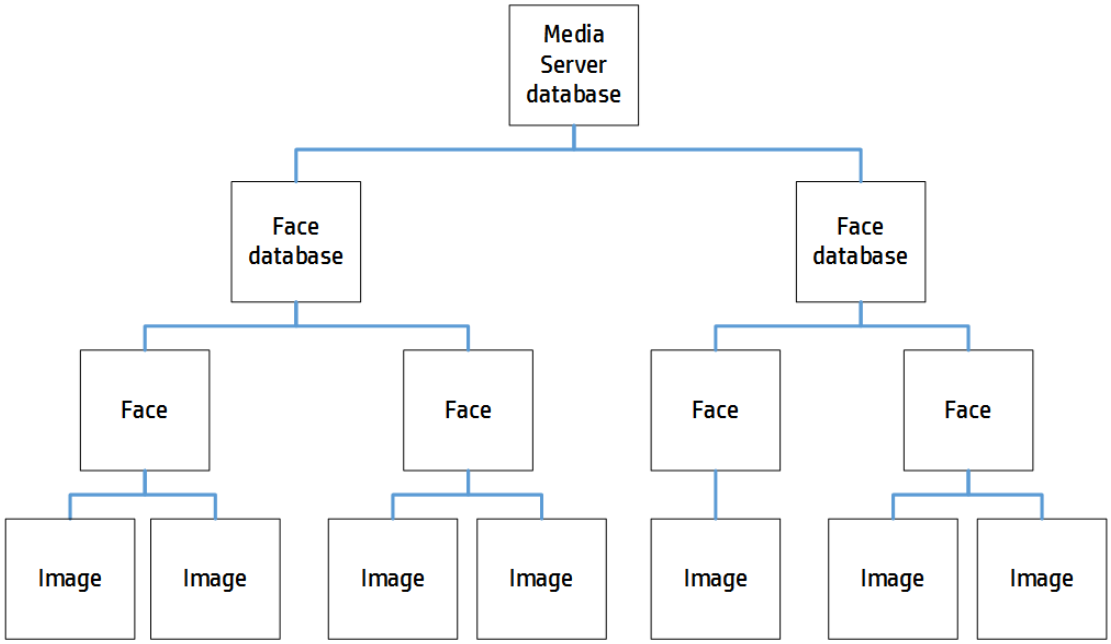
For more information about the parameters that you can use to configure Face Detection, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Train Media Server to Recognize Faces

To run face recognition you must train Media Server by providing images of people that you want to identify. When you run face recognition, Media Server compares the faces it detects in the video to the faces that you added during training.

The following diagram shows how Media Server stores information you provide during training.



The "Media Server database" can represent either the internal Media Server database or a database on an external database server. For information on setting up this database, see "Set up a Database to Store Training Data" on page 29.

You can organize faces into databases. These face databases are for organizational purposes. For example, you might have a database for politicians, a database for actors and a database for musicians. If you want to recognize only actors, you can then run face recognition using only that database.

A database can contain any number of faces, each representing a single person. Each face has an identifier, which must be unique. You can associate metadata with a face, for example the person's name or other information.

To each face you must add one or more training images. HPE recommends adding multiple training images for best performance. For information about choosing suitable training images, see "Select Images for Training" below.

Select Images for Training

Add one or more training images to each face that you want to recognize.

HPE recommends using images that are representative of how you will use face recognition. For example, if you intend to use face recognition in a situation that is not controlled for facial expression, add several images to each face showing different facial expressions.

It is better to use fewer high-quality images, rather than many low-quality images, because low-quality images can reduce accuracy.

A good training image for a face:

- must contain only one face.
- must be rotated correctly (the face cannot be upside down).
- contains a face that is wider than approximately 150 pixels; the face should constitute a large proportion of the image.
- contains a front view of the face, so that both eyes are visible.
- is not blurry.
- has even illumination, because dark shadows or bright highlights on the face reduce recognition accuracy.
- has not been converted from another file format to JPEG. Converting images to JPEG introduces compression artifacts. If you have JPEG images, avoid making many rounds of edits. Each time that a new version of the image is saved as a JPEG, the image quality degrades.
- shows a face that is not partially obscured (for example a person standing behind a microphone).

Create a Database to Contain Faces

To create a database to contain faces, use the following procedure.

To create a database to contain faces

- Use the `CreateFaceDatabase` action with the `database` parameter.

  `database` The name of the new database (maximum 254 bytes).

  For example:

```
curl http://localhost:14000 -F action=CreateFaceDatabase
                            -F database=Faces
```

Add a Face to a Database

To add a face to a database of faces, you can:

- **Perform training in separate steps**. Create a new (empty) face, then add training images, and then train Media Server, using separate actions. You can also add metadata to the face but this is an optional step. You might want to perform training in this way if you are building a front end application that responds to user input.
- **Perform training in a single action**. You might use this approach when writing a script to train Media Server from a collection of images and metadata.

Add a Face to a Database (Using Separate Steps)

This section describes how to create a new (empty) face, then add training images, and then train Media Server, using separate actions. You can also add metadata to the face but this is an optional step. You might want to perform training in this way if you are building a front end application that responds to user input.

Alternatively, you can train Media Server to recognize a face by sending a single action. To do this, see "Add a Face to a Database (Using a Single Action)" on the next page.

To add a face to a database (using separate steps)

1. Add a face using the `NewFace` action. Set the following parameters:

`database`	The name of the database to add the face to. The database must already exist.
`identifier`	(Optional) A unique identifier for the face (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically. You can use the person's name as the identifier, but you might have several people with the same name and all identifiers within a database must be unique. Alternatively, allow Media Server to generate an identifier and add the person's name to a metadata field (see step 3, below).

   For example:

   ```
   curl http://localhost:14000 -F action=NewFace
                               -F database=Faces
   ```

   Media Server adds the face and returns the identifier.

2. Add one or more training images to the face using the `AddFaceImages` action. Set the following parameters:

`database`	The name of the database that contains the face.
`identifier`	The identifier for the face, returned by the `NewFace` action.
`imagedata`	The training images to add. Files must be uploaded as multipart/form-data. For more information about sending data to Media Server, see "Send Data by Using a POST Method" on page 67.

`imagelabels`	A comma-separated list of labels. One label is associated with each image. (maximum 254 bytes for each label). The number of labels must match the number of images. If you do not set this parameter, Media Server generates labels automatically.

For example:

```
curl http://localhost:14000 -F action=AddFaceImages
                            -F database=Faces
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
                            -F imagedata=@face1_smile.jpg,face1_neutral.jpg
                            -F imagelabels=image1,image2
```

3. (Optional) Add metadata to the face using the `AddFaceMetadata` action. You can add any number of key-value pairs. Set the following parameters:

`database`	The name of the database that contains the face.
`identifier`	The identifier for the face, returned by the `NewFace` action.
`key`	The key to add (maximum 254 bytes).
`value`	The value to add (maximum 254 bytes).

For example:

```
curl http://localhost:14000 -F action=AddFaceMetadata
                            -F database=Faces
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
                            -F key=firstname
                            -F value=John
```

4. Complete the training for the face using the `BuildFace` action. Set the following parameters:

`database`	The name of the database that contains the face.
`identifier`	The identifier for the face, returned by the `NewFace` action.

For example:

```
curl http://localhost:14000 -F action=BuildFace
                            -F database=Faces
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
```

Add a Face to a Database (Using a Single Action)

You can train Media Server to recognize a face by sending a single action (`TrainFace`).

Running this action is equivalent to running the following actions in the following order:

- `NewFace`
- `AddFaceImages`
- `AddFaceMetadata` (optional)
- `BuildFace`

The `TrainFace` action is atomic, so that any interruption to the server does not leave the database in an inconsistent state.

Alternatively, you can train Media Server by sending these actions individually. You might want to do this if you are building a front end application that trains Media Server in response to user input. For more information about how to do this, see "Add a Face to a Database (Using Separate Steps)" on page 118.

To add a face to a database (using a single action)

- Add a face using the `TrainFace` action. Set the following parameters:

`database`	The name of the database to add the face to. The database must already exist.
`identifier`	(Optional) A unique identifier for the face (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.
`imagedata`	The training images to add. Files must be uploaded as multipart/form-data. For more information about sending data to Media Server, see "Send Data by Using a POST Method" on page 67.
`imagelabels`	(Optional) A comma-separated list of labels. One label is associated with each image (maximum 254 bytes for each label). The number of labels must match the number of images. If you do not set this parameter, Media Server generates labels automatically.
`metadata`	(Optional) A comma-separated list of metadata key-value pairs to add to the face. Separate keys from values using a colon (:). To include a comma or colon in a key name or value, you must enclose the key name or value in quotation marks (") and escape any quotation marks that occur within the string with a backslash (\).

For example:

```
curl http://localhost:14000 -F action=TrainFace
                            -F database=Faces
                            -F imagedata=@face1_smile.jpg,face1_neutral.jpg
                            -F imagelabels=image1,image2
                            -F metadata=lastname:Smith,fullname:"John Smith, Jr"
```

Media Server adds the face to the database and returns the identifier.

List the Faces in a Database

To list the faces that you have added to a database, and check whether training was successful, use the following procedure.

To list the faces in a database

1. (Optional) First list the databases that have been created to store faces. Use the action `ListFaceDatabases`:

   ```
   http://localhost:14000/action=ListFaceDatabases
   ```

   Media Server returns a list of databases that you have created to store faces.

2. List the faces that exist in one of the databases. Use the action `ListFaces`, for example:

   ```
   http://localhost:14000/action=ListFaces&database=faces
                                          &metadata=true
                                          &imagestatus=true
   ```

   Media Server returns a list of faces in the specified database, and the number of training images associated with each face.

   If you set the action parameter `metadata` to **`true`**, Media Server returns the metadata you have added to a face.

   If you set the action parameter `imagestatus` to **`true`**, Media Server returns the status of each training image associated with each face.

   - A status of `trained` indicates that training was successful.
   - If images have a status of `untrained`, run training for the face using the action `BuildFace`, or run training for all faces that have incomplete training using the action `BuildAllFaces`.
   - If images have a status of `failed`, Media Server could not use the image for training. For example, if Media Server does not detect a face in an image, it cannot be used as a training image. Remove the failed images using the action `RemoveFaceImages`.

Update or Remove Faces and Databases

To update or remove a face use the following actions:

- To complete training for all faces that exist in the Media Server database but have incomplete training, use the action `BuildAllFaces`. To confirm that training was successful, use the action `ListFaces` (see "List the Faces in a Database" on the previous page).
- To add additional training images to a face, use the action `AddFaceImages`. Media Server does not use the new images for face recognition until you run training using the action `BuildFace` or `BuildAllFaces`. To confirm that training was successful, use the action `ListFaces` (see "List the Faces in a Database" on the previous page).
- To remove a training image from a face, for example if Media Server cannot detect a face in the image, use the action `RemoveFaceImages`.
- To change the label of an image that you added to a face, use the action `RelabelFaceImage`.
- To move an image of a face from one face to another, for example if you add an image to the wrong face, use the action `MoveFaceImage`.
- To add, remove, or update custom metadata for a face, use the actions `AddFaceMetadata`, `RemoveFaceMetadata`, and `UpdateFaceMetadata`.
- To change the identifier of a face you added to a face database, use the action `RenameFace`.
- To remove a face from a database and discard all associated images and metadata, use the action `RemoveFace`.

To update or remove face databases, use the following actions:

- To rename a face database, use the action `RenameFaceDatabase`.
- To delete a face database and all of the information that it contains, use the action `RemoveFaceDatabase`.

For more information about the actions that you can use to train Media Server, refer to the *Media Server Reference*.

Synchronize with the Latest Training

Training Media Server updates the training database. However, the training is not used in analysis operations until Media Server has synchronized with the database. Synchronization loads the data from the database into memory and makes it ready for use. Synchronization is necessary because several Media Servers can share a single database, and you can use any Media Server to perform training.

The default configuration ensures that analysis uses the latest training data, but you can customize Media Server for your use case.

Automatic Synchronization

The default configuration ensures that analysis uses the latest training data, no matter how often you train Media Server and how much training data you add to the database.

Before it starts a `process` action that requires training data, Media Server synchronizes with the training database and waits for the synchronization to complete. This is because the `SyncDatabase` parameter - available for face recognition, object recognition, and image classification tasks - has a default value of `TRUE`.

Media Server also synchronizes with the database at regular intervals (by default, every second). You can configure the interval between each synchronization by setting the `SyncInterval` parameter in the `[Database]` section of the configuration file. This is important for processing video streams, because when you process a stream Media Server could continue running the same action for days or weeks. Automatic synchronization ensures that Media Server regularly loads the latest training without you having to intervene.

Manual Synchronization

In some cases, you might want to control when Media Server synchronizes with the training database:

- In a production environment, you might modify the training only at known times. In this case you might prefer to disable automatic synchronization, because forcing Media Server to query the database before it starts processing each request can reduce the processing speed.
- You might want to disable automatic synchronization to ensure that a batch of `process` requests all use the same training. If you are processing discrete files, you might perform training in bulk and then process batches of files.

To change the frequency at which Media Server automatically synchronizes with the database, or disable scheduled synchronization completely, set the parameter `SyncInterval` in the `[Database]` section of the configuration file.

To prevent Media Server synchronizing with the database before it starts a `process` action, set `SyncDatabase=False` in the relevant analysis tasks. With this setting, be aware that `process` requests could use outdated training. This is especially likely if you add large amounts of training data to the database and then immediately start a `process` action, because Media Server might not have finished training before it starts processing.

If you disable automatic synchronization but want to ensure that Media Server has synchronized with the latest training, run the `SyncFaces`, `SyncObjects`, and `SyncClassifiers` actions before you send the `process` action. These actions force Media Server to synchronize with the database. After they complete, you can be sure that Media Server is using the latest training data.

Reduce Memory Use

With automatic synchronization, described above, Media Server loads all training data into memory. If you have an extremely large training database that contains many face databases, object databases, and classifiers, this can consume a significant amount of memory.

You can remove all face databases, object databases, or classifiers from memory by setting `SyncInterval=0` (to disable the automatic synchronization) and running the action `UnsyncFaces`, `UnsyncObjects`, or `UnsyncClassifiers`.

You can then load the training you need using one of the following methods:

- In the `[TaskName]` section for your analysis tasks, set `SyncDatabase=TRUE`. Media Server will load the training it needs to complete the tasks.
- In the `[TaskName]` section for your analysis task, set `SyncDatabase=FALSE`. Then, manually load the training data you need by running the actions `SyncFaces`, `SyncObjects`, and `SyncClassifiers`. You can add the `database` parameter to the `SyncFaces` and `SyncObjects` actions to choose which database to load into memory.

IDOL Admin

You can train Media Server using IDOL Admin. To open the IDOL Admin interface, send the following action to Media Server's ACI port:

```
http://localhost:14000/action=admin
```

For more information about using IDOL Admin refer to the *IDOL Admin User Guide*.

Recognize Faces

This section describes how to create an analysis task to recognize faces that appear in media.

Tip: To run face recognition, you must first detect faces by setting up a face detection task. For information about how to do this, see "Detect Faces" on page 115.

To recognize faces in media

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

```
[Analysis]
AnalysisEngine0=FaceDetect
AnalysisEngine1=FaceRecognize
```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`FaceRecognize`**.
`Input`	The track that contains detected faces that you want to recognize. Set this parameter to the `ResultWithSource` output track from your face detection task. For example, if your face detection task is named `FaceDetect`, set this parameter to `FaceDetect.ResultWithSource`.
`RecognitionThreshold`	(Optional) The minimum confidence score required to recognize a face.
`MaxRecognitionResults`	(Optional) The maximum number of results to return, if the face matches more than one entry in the training database(s).
`Database`	(Optional) The database to use for recognizing the detected faces. By default, Media Server uses all available data. Database names are case-sensitive.

   For example:

```
[FaceRecognize]
Type=FaceRecognize
Input=FaceDetect.ResultWithSource
RecognitionThreshold=60
MaxRecognitionResults=1
```

   For a complete list of parameters that you can use to configure a face recognition task, and more information about these parameters, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Obtain Demographic Information

This section describes how to obtain demographic information for detected faces.

Tip: You must first detect faces by setting up a face detection task. For information about how to do this, see "Detect Faces" on page 115.

To obtain demographic information for detected faces

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter.

You can give the task any name, for example:

```
[Analysis]
AnalysisEngine0=FaceDetect
AnalysisEngine1=FaceDemographics
```

3. Create a new configuration section to contain the task settings and set the following parameters:

   `Type` The analysis engine to use. Set this parameter to **`Demographics`**.

   `Input` The track that contains detected faces that you want to analyze. Set this parameter to the `ResultWithSource` output track from your face detection task. For example, if your face detection task is named `FaceDetect`, set this parameter to `FaceDetect.ResultWithSource`.

   For example:

   ```
   [FaceDemographics]
   Type=demographics
   Input=FaceDetect.ResultWithSource
   ```

   For more information about these parameters, refer to the *Media Server Reference*.
4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Analyze Facial Expression

This section describes how to analyze the facial expression for detected faces.

Tip: You must first detect faces by setting up a face detection task. For information about how to do this, see "Detect Faces" on page 115.

Note: If the person has only one eye open (for example, if they are winking), Media Server reports that their eyes are open.

To analyze facial expression

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=FaceDetect
   AnalysisEngine1=FaceState
   ```
3. Create a new configuration section to contain the task settings, and set the following parameters:

   `Type` The analysis engine to use. Set this parameter to **`FaceState`**.

`Input` The track that contains detected faces that you want to analyze. Set this parameter to the `ResultWithSource` output track from your face detection task. For example, if your face detection task is named `FaceDetect`, set this parameter to `FaceDetect.ResultWithSource`.

For example:

```
[FaceState]
Type=facestate
Input=FaceDetect.ResultWithSource
```

For more information about these parameters, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Find Clothing

Clothing analysis provides the location of the clothing (covering the upper body) of a person who has been identified by face detection or face recognition.

To determine the location of clothing

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

```
[Analysis]
AnalysisEngine0=FaceDetect
AnalysisEngine1=Clothing
```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type` The analysis engine to use. Set this parameter to **`Clothing`**.

`Input` The track that contains detected faces. The clothing analysis engine provides the position of a person's clothing for each detected face. Set this parameter to the `ResultWithSource` output track from your face detection task. For example, if your face detection task is named `FaceDetect`, set this parameter to `FaceDetect.ResultWithSource`.

For example:

```
[Clothing]
Type=Clothing
FaceInput=FaceDetect.ResultWithSource
```

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

You can run clothing analysis followed by a color analysis task to find the dominant color of a person's clothing. For more information about the color analysis task, see "Analyze Colors" on page 193. The following procedure describes how to add a task for color analysis.

To determine the color of clothing

1. Open the configuration file in which you configured the clothing analysis task.
2. In the `[Analysis]` section, add a new analysis task (for color analysis) by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=FaceDetect
   AnalysisEngine1=Clothing
   AnalysisEngine2=ClothingColors
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`ColorCluster`**.
`Input`	The track that contains information about the location of the clothing (and the image to analyze). Set this parameter to the `ResultWithSource` output track from your clothing analysis task. For example, if your clothing analysis task is named `Clothing`, set this parameter to `Clothing.ResultWithSource`.
`RestrictToInputRegion`	A Boolean value that specifies whether to analyze a region of the input image or video frame that is specified in the input record, instead of the entire image. Set this parameter to **`TRUE`**, because you want the color analysis task to analyze only the region that represents the clothing, and not the entire image.

   For example:

   ```
   [ClothingColors]
   Type=ColorCluster
   Input=Clothing.ResultWithSource
   RestrictToInputRegion=True
   ```

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Face Detection Results

The following XML shows a single record produced by face detection.

```
<output>
  <record>
    ...
    <trackname>FaceDetect.Result</trackname>
    <FaceResult>
      <id>4895aeee-6a8f-44f9-915c-b86eff702118</id>
      <face>
```

```
        <region>
          <left>282</left>
          <top>84</top>
          <width>236</width>
          <height>236</height>
        </region>
        <outOfPlaneAngleX>0</outOfPlaneAngleX>
        <outOfPlaneAngleY>0</outOfPlaneAngleY>
        <ellipse>
          <center>
            <x>398.5</x>
            <y>194.25</y>
          </center>
          <a>106.25</a>
          <b>148.75</b>
          <angle>0</angle>
        </ellipse>
        <lefteye>
          <center>
            <x>441</x>
            <y>173</y>
          </center>
          <radius>16</radius>
        </lefteye>
        <righteye>
          <center>
            <x>356</x>
            <y>173</y>
          </center>
          <radius>16</radius>
        </righteye>
      </face>
    </FaceResult>
  </record>
</output>
```

The record contains the following information:

- The `id` element provides a unique identifier for the detected face. The face detection engine issues an ID for each detected appearance of a face. If you are detecting faces in video and consecutive frames show the same face in a near-identical location, all records related to that appearance will have the same ID.

  For example, if a face appears in the same location for a hundred consecutive video frames, the engine uses the same ID for each record in the data track and the single record in the result track. The record in the result track will have a timestamp that covers all of the frames.

  If the face disappears and then reappears, the engine considers this as a new detection and produces a new ID and a new record in the result track.
- The `face` element contains the location of the detected face:

- `region` describes the location of the face within the image or video frame. The `left` and `top` elements provide the position of the top-left corner of a rectangle that surrounds the face, and the `width` and `height` elements provide its size.
- `outOfPlaneAngleX` indicates how far the person is looking to the left or right. `outOfPlaneAngleY` indicates how far the person is looking up or down. When both of these angles are zero, the person is looking directly at the camera.
- the `ellipse` element describes the location of the detected face as a circle or ellipse. When `DetectEyes=FALSE`, Media Server returns a circle that describes the approximate position of the face. When `DetectEyes=TRUE` and the person is looking directly at the camera, Media Server returns an ellipse that should more accurately describe the position.
- the `lefteye` and `righteye` elements describe eye locations. These are returned only if `DetectEyes=TRUE` and the person is looking directly towards the camera.

Face Recognition Results

The following XML shows a single record produced by face recognition.

```
<record>
  ...
  <trackname>facerec.Result</trackname>
  <FaceRecognitionResult>
    <id>69ff21d8-bc3b-44b1-9a47-1eabedb75dd0</id>
    <face>
      ...
    </face>
    <identity>
      <identifier>BarackObama</identifier>
      <database>politicians</database>
      <imagelabel>image1</imagelabel>
      <confidence>50.11</confidence>
    </identity>
  </FaceRecognitionResult>
</record>
```

The record contains the following information:

- The `id` element contains a unique identifier for the detected face. The face detection engine issues an ID for each detected appearance of a face, and the records output from the face recognition engine use the same IDs as the input records.
- The `face` element contains the location of the detected face. For more information about this data, see "Face Detection Results" on page 127.
- Each `identity` element represents a match between the detected face and a face in your training database.
  - The `identifier` element provides the identifier of the database entry that matched the face.
  - The `database` element provides the name of the database in which the match was found.

- The `imagelabel` element provides the label of the image that was the best match to the detected face.
- The `confidence` element provides the confidence score for the match.

Face Demographics Results

The following XML shows a single record produced by face demographics analysis.

```
<record>
  ...
  <trackname>FaceDemographics.Result</trackname>
  <DemographicsResult>
    <id>8774d02a-dcf0-4410-b591-bd2b7d3981f5</id>
    <face>
      ...
    </face>
    <ethnicity>Caucasian</ethnicity>
    <age>Elderly</age>
    <gender>Male</gender>
  </DemographicsResult>
</record>
```

The record contains the following information:

- The `id` element contains a unique identifier for the detected face. The face detection engine issues an ID for each detected appearance of a face, and the records output from the face demographics engine use the same IDs as the input records.
- The `face` element contains the location of the detected face. For more information about this data, see "Face Detection Results" on page 127.
- The `ethnicity` element provides the ethnicity of the person:
  - `African/Caribbean`
  - `Arab`
  - `Caucasian`
  - `East Asian`
  - `Hispanic`
  - `Indian Subcontinent`
- The `age` element provides the approximate age of the person, as one of the following values:
  - `Baby` (below approximately 2 years)
  - `Child` (approximately 2–15 years)
  - `Young Adult` (approximately 15–35 years)

- `Adult` (approximately 35–55 years)
- `Elderly` (above approximately 55 years)

- The `gender` element provides the gender of the person (either `Male` or `Female`). However, Media Server does not attempt to determine the gender of babies, and will return the value `Unclassified`.

Face Expression Analysis Results

The following XML shows a single record produced by face expression analysis.

```
<record>
  ...
  <trackname>FaceExpression.Result</trackname>
  <FaceStateResult>
    <id>ca444959-3609-4e06-a633-1bd95e7b3440</id>
    <face>
      ...
    </face>
    <expression>Neutral</expression>
    <eyesopen>true</eyesopen>
    <spectacles>false</spectacles>
  </FaceStateResult>
</record>
```

The record contains the following information:

- The `id` element contains a unique identifier for the detected face. The face detection engine issues an ID for each detected appearance of a face, and the records output from the face state analysis engine use the same IDs as the input records.
- The `face` element contains the location of the detected face. For more information about this data, see "Face Detection Results" on page 127.
- The `expression` element describes the facial expression (either `Happy` or `Neutral`).
- The `eyesopen` element specifies whether the person's eyes are open (`true`) or closed (`false`). If the person has only one eye open (for example, if they are winking), Media Server reports that their eyes are open.
- The `spectacles` element specifies whether the person is wearing spectacles.

Clothing Analysis Results

The following XML shows a single record produced by clothing analysis.

```
<record>
   ...
   <trackname>Clothing.Result</trackname>
   <ClothingResult>
     <id>63f77e13-e916-4528-acfb-559e1db28574</id>
     <region>
```

```
        <left>116</left>
        <top>190</top>
        <width>248</width>
        <height>480</height>
      </region>
    </ClothingResult>
</record>
```

The record contains the following information:

- The `id` element contains a unique identifier for the detected face. The face detection engine issues an ID for each detected appearance of a face, and the records output from the clothing analysis engine use the same IDs as the input records.
- The `region` element contains the location of the clothing covering the person's upper body. The values are in pixels. The `left` and `top` elements give the position of the top-left corner of a rectangle, and `width` and `height` provide its width and height.

Optimize Face Analysis Performance

The quality of the image or video that you send to Media Server can have a significant effect on the performance of face analysis.

- In order to be detected, faces must exceed the minimum size specified by the `MinSize` parameter in your task configuration.
- In order to recognize faces and run demographics analysis and state analysis, the face must be large enough in the image that facial features are clearly visible.
- Faces should be in focus, not blurry.
- Face detection performs best when faces are looking towards the camera, so that both eyes are visible, but faces can be detected when viewed in profile (side-on). For face recognition, demographics analysis, and expression analysis, the person must be looking toward the camera so that both eyes are visible.
- Ideally faces should be fully visible and not be partially obscured (for example a person standing behind a microphone).
- Although face detection can process a relatively wide range of facial expressions, faces with neutral expressions are usually detected with the greatest reliability. Particularly unusual facial expressions might cause face detection and recognition to fail.
- Spectacles or large amounts of facial hair increase the difficulty in detecting and recognizing faces and accuracy may be reduced in these cases.
- The image or video should have even illumination, because dark shadows or bright highlights on the face reduce accuracy.
- The image or video should not be affected by significant compression artifacts that can affect some formats (such as highly compressed JPEG images). HPE recommends that you do not save your image files as JPEG images with high levels of compression or transcode video that has been captured from a camera. If you are using a digital video recorder (DVR) to record the footage from a camera and are then sending the video to Media Server, ensure the DVR is saving the video at full resolution and is not reducing the bit rate.

Chapter 11: Perform Optical Character Recognition

Media Server can perform Optical Character Recognition (OCR) to extract text from media. OCR makes text in images and video computer-readable, so that it can be indexed into IDOL Server and used in analysis operations.

Introduction

Media Server can run Optical Character Recognition (OCR) on images such as scanned documents and photographs of documents. You can also run OCR on video to extract subtitles and scrolling text that sometimes appears during television news broadcasts.

Media Server OCR:

- searches images and video for text-like regions, and only performs OCR on those regions.
- provides options to restrict the language and character types used during recognition, which can increase the accuracy of OCR in some cases.
- supports specialized font types.
- supports many languages.
- can automatically adjust when scanned documents are rotated by either 90 or 180 degrees from upright.
- can automatically adjust for skewed text in scanned documents and photographs.

Note: Media Server OCR recognizes machine-printed text. Handwritten text is not supported.

OCR Document File Formats

When you ingest a PDF or office document file, Media Server extracts both embedded images and text elements. The OCR engine runs OCR on the images that are extracted from the document, and by default, merges the text that was contained in text elements into the results. This means that the OCR results contain both the text that is extracted from images and the text that was contained in text elements.

Set up an OCR Analysis Task

To perform OCR

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*N* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=OCR
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`OCR`**.
`Input`	(Optional) The track to process. If you do not specify an input track, Media Server processes the first track of the correct type that is produced by the ingest engine.
`Languages`	The language of the text. Setting this parameter narrows the character and word choices for the OCR process, which increases speed and accuracy. For a list of supported languages, see "OCR Supported Languages" on page 276.
`ProcessTextElements`	(Optional) (Only affects documents) Specifies whether to merge the content of text elements into the OCR results. If the text elements in the document are not consistent with the text that appears in the image, you might want to set this parameter to **`false`**.
`CharacterTypes`	(Optional) If the document uses a particular type of characters only, such as all uppercase, or all lowercase, you can specify the type in the `CharacterTypes` parameter. This can improve accuracy.
`HollowText`	(Optional) Set this parameter if you are processing subtitles that consist of white characters with black outlines.
`Region`	(Optional) Restrict OCR to a region of the image, instead of the entire image.
`RegionUnit`	(Optional) The units to use for specifying the region (default `pixel`). To specify the position and size of the region as a percentage of the media size, set this parameter to **`percent`**.

   For example:

   ```
   [OCR]
   Type=ocr
   Languages=en
   ```

For more information about the parameters you can use to customize OCR, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

OCR Results

This section describes the format of the results produced by an OCR analysis task.

Results by Line

The following XML shows records from the `Result` track of an OCR task. The analysis engine produces one record for each line of text in the analyzed image or video frame.

If you are processing a document, then unless you have set `ProcessTextElements=FALSE`, some of the records in the `Result` track could represent text that has been extracted from text elements that were present in the document.

```
<record>
  ...
  <trackname>ocr.Result</trackname>
  <OCRResult>
    <id>14565401-b521-4135-94c8-b30f02264f38</id>
    <text>rover discovers life on Mars</text>
    <region>
      <left>240</left>
      <top>31</top>
      <width>194</width>
      <height>12</height>
    </region>
    <confidence>89</confidence>
    <angle>0</angle>
    <source>image</source>
  </OCRResult>
</record>
<record>
  ...
  <trackname>ocr.Result</trackname>
  <OCRResult>
    <id>59dad245-c268-4506-ac42-5752dd123576</id>
    <text>discovery confirmed yesterday and announced to world press</text>
    <region>
      <left>120</left>
      <top>62</top>
      <width>434</width>
      <height>15</height>
    </region>
    <confidence>88</confidence>
```

```
    <angle>0</angle>
    <source>image</source>
  </OCRResult>
</record>
```

Each record contains the following information:

- The `id` element provides a unique identifier for the line of text. The OCR analysis engine issues an ID for each detected appearance of a line of text. If you are running OCR on video and consecutive frames show the same text, all records related to that appearance will have the same ID.

  For example, if text appears in the same location for fifty consecutive video frames, the engine uses the same ID for each record in the data track and produces a single record in the result track. The record in the result track will have a timestamp that covers all fifty frames.

  If the text moves to a different location on the screen, or disappears and then reappears, the engine considers this as a new detection and produces a new ID and a new record in the result track.
- The `text` element contains the text recognized by OCR.
- The `region` element describes the position of the text in the ingested media. If the record represents a text element that has been extracted from a document, the region is accurate only if the source media was a PDF file. Position information is not extracted from other document formats.
- The `confidence` element provides the confidence score for the OCR process (from 0 to 100). For text that was extracted from a text element in a document, the confidence score is always `100`.
- The `angle` element gives the orientation of the text (rotated clockwise in degrees from upright).
- The source element specifies whether the text originated from an image or video frame (`image`) or a text element in a document (`text`).

Results by Word

An OCR analysis task that analyzes an image or document (but not video) also produces a `WordResult` output track. To this track the OCR analysis engine adds a record for each word. The following XML shows an example record.

Note: Text that is extracted from a text element in a document is not output to the `WordResult` track.

```
<record>
  ...
  <trackname>ocr.WordResult</trackname>
  <OCRResult>
    <id>cdbca09b-c289-40af-b6e6-02427fafad91</id>
    <text>rover</text>
    <region>
      <left>240</left>
      <top>31</top>
      <width>194</width>
      <height>12</height>
    </region>
    <confidence>89</confidence>
    <angle>0</angle>
```

```
    <source>image</source>
  </OCRResult>
</record>
```

Improve OCR

To get the best results from OCR, the ingested images or video must have sufficient contrast and be in focus.

The minimum height for a readable character in a high-quality image is approximately 10 pixels. In a poorer quality image, the characters must be larger.

If you know that all of the text in your images matches a specific character type, for example is upper case or only contains numbers, set the `CharacterTypes` configuration parameter. This can improve accuracy in some cases, because Media Server does not look for other character types.

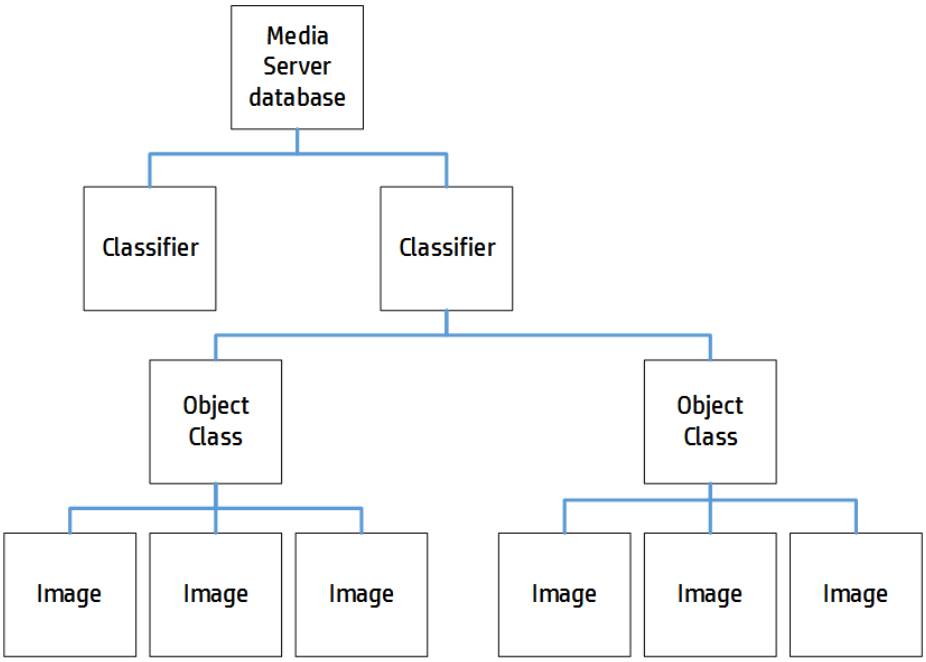
Chapter 12: Classify Images

Image classification classifies images or video frames. For example, you could create a classifier named `vehicles` and train Media Server to classify each vehicle detected by Number Plate Recognition or Scene Analysis as a car, truck, or motorcycle.

Train Media Server to Classify Images

To classify images, you must train Media Server by providing images that are representative of your chosen classes.

The following diagram shows how Media Server stores information you provide during training.



The "Media Server database" represents the Media Server datastore file or a database on an external database server. For information about how to set up this database, see "Set up a Database to Store Training Data" on page 29.

A classifier contains the object classes that you want to use for a classification task. When you run image classification, Media Server classifies your images into the classes in your chosen classifier. For example, to classify vehicles into "cars", "trucks", "motorcycles", and so on, you would create a classifier named "vehicles" and create object classes named "car", "truck", and "motorcycle".

A classifier must contain at least two object classes. To determine whether an image contains a car or doesn't contain a car, you would need to train a "car" object class and also a "not car" object class.

To each object class you must add training images. Usually around 100 training images per object class is sufficient to obtain good performance. For information about choosing suitable training images, see "Training Requirements" on the next page.

Classifier Types

Media Server supports several types of image classifier.

HPE recommends trying each type of classifier to determine which produces the best results for your purposes. When you test your classifier, ensure that the images you use for testing are different to your training images.

Bayesian

The default type of classifier is the `Bayesian` classifier. Use a `Bayesian` classifier when you have a small number of classes and many training images for each class.

The `Bayesian` classifier is better than the `Maxvote` classifier at distinguishing between classes that have a high degree of similarity, and requires less time to train than the `CNN` classifier.

When you run classification, the `Bayesian` classifier outputs a confidence score for each class. These scores can be compared across classifiers, and you can set a threshold to discard results below a specified confidence level.

This is the only type of classifier that supports prior probabilities.

Maxvote

Use a `Maxvote` classifier when you have many classes with fewer training images per class. This type of classifier requires fewer training images and can be trained faster than either the `Bayesian` or `CNN` classifier.

When you run classification, the `Maxvote` classifier outputs a number of votes for each class in the classifier. This means that you cannot compare scores across classifiers.

CNN

The Convolutional Neural Network (`CNN`) classifier usually produces the most accurate results, but can require a significant amount of time to train.

The more time you allow Media Server to train the classifier, the greater the accuracy. Before you train a `CNN` classifier, you can choose how many training iterations to run. The time required to train the classifier is proportional to the number of training iterations and the number of training images. Increasing the number of iterations always improves the training and results in better accuracy, but each additional iteration that you add has a smaller effect.

For classifiers that have four or five dissimilar classes with around 100 training images per class, approximately 500 iterations produces reasonable results. This number of iterations with this number of training images requires approximately five hours to complete. HPE recommends a larger number of

iterations for classifiers that contain many similar classes. For extremely complex classifiers that have hundreds of classes, you might run 200,000 training iterations. Be aware that running this number of training iterations with large numbers of training images is likely to take weeks.

To find the optimum number of iterations, HPE recommends that you start with a small number of iterations. Double the number of iterations each time you train, until classification accuracy is acceptable.

When you run classification, the `CNN` classifier outputs a confidence score for each class. These scores can be compared across classifiers, and you can set a threshold to discard results below a specified confidence level.

Training Requirements

The performance of classification is generally better if:

- the classifier contains only a few object classes (but it must contain at least two classes).
- the object classes are dissimilar. For example, when training a 'field' class and a 'beach' class, the presence of clouds in the sky in both sets of training images might cause confusion between the classes.
- the object classes are trained with many images. Usually around 100 images are sufficient to train an object class. If the images in a class are very similar, fewer images might be sufficient.
- the training images are representative of the variation typically found within the object class. For example, to train a "dog" class, use images of dogs of different sizes, breeds, colors, and from different viewpoints.
- the training images contain little background or clutter around the object in the image.
- the longest dimension (width or height) of the training image is at least 500 pixels - smaller images might result in reduced accuracy.

  Tip: High-resolution images where the object covers a small proportion of the image make poor training images. If you have a large image showing the object and it can be cropped such that its longest dimension still exceeds 500 pixels, HPE recommends cropping the image. If you crop an image, leave a gap around the object of at least 16 pixels.

Create a Classifier

A classifier contains object classes. When you run classification, Media Server uses a single classifier and classifies images into the classes in that classifier.

To create a classifier

1. Use the `CreateClassifier` action with the `classifier` parameter.

   `classifier` The name of the new classifier (maximum 254 bytes).

   For example:

   ```
   curl http://localhost:14000 -F action=CreateClassifier
                               -F classifier=vehicles
   ```

2. (Optional) Set training options on the classifier (such as selecting the type of classifier) using the

`SetClassifierTrainingOption` action:

`classifier`	The name of the classifier that you created in the previous step.
`key`	The name of the training option to set.
`value`	The value for the training option.

For example:

```
curl http://localhost:14000 -F action=SetClassifierTrainingOption
                            -F classifier=vehicles
                            -F key=classifierType
                            -F value=maxvote
```

Classifier Training Options

After you create a new classifier, you can set training options for it.

Set or modify training options using the actions `SetClassifierTrainingOption` and `UnsetClassifierTrainingOption`. When you change a training option all training associated with the classifier becomes invalid and you must retrain the classifier using the `BuildClassifier` action. For more information about these actions, refer to the *Media Server Reference*.

You can set the following training options:

Training Option	Description	Default value
`classifierType`	The type of classifier that you want to train: • **`Bayesian`** • **`Maxvote`** • **`CNN`** - Convolutional Neural Network. For information about each type of classifier, see "Classifier Types" on page 139	Bayesian
`iterations`	The number of iterations to perform when training a neural network classifier. For information about how to set this training option, see "Classifier Types" on page 139.	500

For an example that shows how to add a new classifier and set training options, see "Create a Classifier" on the previous page.

Add Classes to a Classifier

To add object classes to a classifier, complete the following procedure.

To add classes to a classifier

1. Add each new class using the action `CreateClass`. Set the following parameters:

`classifier`	The name of the classifier to add the object class to. The classifier must already exist. To create a classifier, see "Create a Classifier" on page 140.
`identifier`	(Optional) A unique identifier for the object class (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.
`imagedata`	The training images to add. Files must be uploaded as multipart/form-data. For more information about sending data to Media Server, see "Send Data by Using a POST Method" on page 67.
`imagelabels`	(Optional) A comma-separated list of labels. One label is associated with each image. (maximum 254 bytes for each label). The number of labels must match the number of images. If you do not set this parameter, Media Server generates labels automatically.

   For example:

   ```
   curl http://localhost:14000 -F action=createclass
                               -F classifier=vehicles
                               -F identifier=cars
                               -F imagedata=@car1.jpg,car2.jpg,...
   ```

   Media Server adds the new class.

2. (Optional) Add metadata to the class using the `AddClassMetadata` action. You can add any number of key-value pairs. Set the following parameters:

`classifier`	The name of the classifier that contains the class.
`identifier`	The identifier for the object class, as returned by the `CreateClass` action.
`key`	The key to add (maximum 254 bytes).
`value`	The value to add (maximum 254 bytes).

   For example:

   ```
   curl http://localhost:14000 -F action=AddClassMetadata
                               -F classifier=vehicles
                               -F identifier=cars
                               -F key=notes
                               -F value=motor%20vehicles
   ```

3. Complete the training for the classifier by running the action `BuildClassifier`:

   ```
   curl http://localhost:14000 -F action=BuildClassifier
                               -F classifier=vehicles
   ```

   This action is asynchronous, so Media Server returns a token. You can use the token with the `QueueInfo` action to retrieve the response.

List Classifiers and Object Classes

To list the classifiers that you have created, and check their status (whether they need training), use the following procedure.

To list classifiers

- Run the action `ListClassifiers`:

  ```
  http://localhost:14000/action=ListClassifiers&trainingoptions=TRUE
  ```

  Media Server returns a list of classifiers that you have created. For example:

  ```
  <autnresponse>
    <action>LISTCLASSIFIERS</action>
    <response>SUCCESS</response>
    <responsedata>
      <classifier>
        <classifier>vehicles</classifier>
        <state>STALE</state>
        <numclasses>3</numclasses>
        <trainingoptions>
          <trainingoption>
            <key>classifiertype</key>
            <value>bayesian</value>
          </trainingoption>
        </trainingoptions>
      </classifier>
    </responsedata>
  </autnresponse>
  ```

  The `state` element specifies whether the classifier needs training. If this element contains the value "stale", train the classifier using the action `BuildClassifier`. The other possible states are `training`, `trained`, and `failed`.

  The `numclasses` element specifies the number of object classes in the classifier.

  If you set the `trainingoptions` parameter to `true`, the response also includes training options that you have set for the classifier.

To list the object classes in a classifier

- Run the action `ListClasses`:

  ```
  http://localhost:14000/action=ListClasses&classifier=vehicles
                                           &metadata=true
                                           &imagelabels=true
  ```

  Media Server returns a list of object classes in the specified classifier.

  If you set the action parameter `imagelabels` to **`true`**, Media Server returns the labels of images associated with each object class.

If you set the action parameter `metadata` to **`true`**, Media Server returns the metadata you have added to each object class.

Update or Remove Object Classes and Classifiers

To update or remove an object class use the following actions:

- To add additional training images to an object class, use the action `AddClassImages`. After adding images to an object class you must retrain the classifier using the action `BuildClassifier`. To confirm that training was successful, use the action `ListClassifiers` (see "List Classifiers and Object Classes" on the previous page).
- To remove training images from an object class, use the action `RemoveClassImages`. After removing images from an object class you must retrain the classifier using the action `BuildClassifier`. To confirm that training was successful, use the action `ListClassifiers` (see "List Classifiers and Object Classes" on the previous page).
- To modify the training options for a classifier, use the actions `SetClassifierTrainingOption` and `UnsetClassifierTrainingOption`.
- To change the label of an image that you added to an object class, use the action `RelabelClassImage`.
- To move an image from one object class to another, for example if you add an image to the wrong object class, use the action `MoveClassImage`.
- To add, remove, or update custom metadata for an object class, use the actions `AddClassMetadata`, `RemoveClassMetadata`, and `UpdateClassMetadata`.
- To rename an object class, use the action `RenameClass`.
- To remove an object class from a classifier and discard all associated images and metadata, use the action `RemoveClass`.

To update or remove a classifier, use the following actions:

- To rename a classifier, use the action `RenameClassifier`.
- To delete a classifier and all of the information that it contains, use the action `RemoveClassifier`.

For more information about the actions that you can use to train Media Server, refer to the *Media Server Reference*.

Synchronize with the Latest Training

Training Media Server updates the training database. However, the training is not used in analysis operations until Media Server has synchronized with the database. Synchronization loads the data from the database into memory and makes it ready for use. Synchronization is necessary because several Media Servers can share a single database, and you can use any Media Server to perform training.

The default configuration ensures that analysis uses the latest training data, but you can customize Media Server for your use case.

Automatic Synchronization

The default configuration ensures that analysis uses the latest training data, no matter how often you train Media Server and how much training data you add to the database.

Before it starts a `process` action that requires training data, Media Server synchronizes with the training database and waits for the synchronization to complete. This is because the `SyncDatabase` parameter - available for face recognition, object recognition, and image classification tasks - has a default value of `TRUE`.

Media Server also synchronizes with the database at regular intervals (by default, every second). You can configure the interval between each synchronization by setting the `SyncInterval` parameter in the `[Database]` section of the configuration file. This is important for processing video streams, because when you process a stream Media Server could continue running the same action for days or weeks. Automatic synchronization ensures that Media Server regularly loads the latest training without you having to intervene.

Manual Synchronization

In some cases, you might want to control when Media Server synchronizes with the training database:

- In a production environment, you might modify the training only at known times. In this case you might prefer to disable automatic synchronization, because forcing Media Server to query the database before it starts processing each request can reduce the processing speed.
- You might want to disable automatic synchronization to ensure that a batch of `process` requests all use the same training. If you are processing discrete files, you might perform training in bulk and then process batches of files.

To change the frequency at which Media Server automatically synchronizes with the database, or disable scheduled synchronization completely, set the parameter `SyncInterval` in the `[Database]` section of the configuration file.

To prevent Media Server synchronizing with the database before it starts a `process` action, set `SyncDatabase=False` in the relevant analysis tasks. With this setting, be aware that `process` requests could use outdated training. This is especially likely if you add large amounts of training data to the database and then immediately start a `process` action, because Media Server might not have finished training before it starts processing.

If you disable automatic synchronization but want to ensure that Media Server has synchronized with the latest training, run the `SyncFaces`, `SyncObjects`, and `SyncClassifiers` actions before you send the `process` action. These actions force Media Server to synchronize with the database. After they complete, you can be sure that Media Server is using the latest training data.

Reduce Memory Use

With automatic synchronization, described above, Media Server loads all training data into memory. If you have an extremely large training database that contains many face databases, object databases, and classifiers, this can consume a significant amount of memory.

You can remove all face databases, object databases, or classifiers from memory by setting `SyncInterval=0` (to disable the automatic synchronization) and running the action `UnsyncFaces`, `UnsyncObjects`, or `UnsyncClassifiers`.

You can then load the training you need using one of the following methods:

- In the `[TaskName]` section for your analysis tasks, set `SyncDatabase=TRUE`. Media Server will load the training it needs to complete the tasks.
- In the `[TaskName]` section for your analysis task, set `SyncDatabase=FALSE`. Then, manually load the training data you need by running the actions `SyncFaces`, `SyncObjects`, and `SyncClassifiers`. You can add the `database` parameter to the `SyncFaces` and `SyncObjects` actions to choose which database to load into memory.

Import a Classifier

HPE may provide classifiers that you can use with Media Server to classify images. These classifiers are pre-trained, and to use one you only need to import the training data into your Media Server database.

For a list of the available pre-trained classifiers, see "Pre-Trained Classifiers" on page 283.

To import a classifier

1. Go to https://downloads.autonomy.com/productDownloads.jsp and download the classifier to your Media Server machine.
2. Import the training data by running the action `ImportClassifer`, for example if you downloaded the file to the folder `pretrained/objectclass`, in the Media Server installation folder:

   ```
   /action=ImportClassifier&classifier=imagenet
                           &classifierpath=./pretrained/objectclass/imagenet.dat
   ```

   where,

`classifier`	The name to give to the imported classifier.
`classifierpath`	The path of the classifier data file on disk
`classifierdata`	The classifier data (you can set this as an alternative to `classifierpath`).

   Media Server imports the data. You can now run classification.

Classify Images

To classify images or video frames

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*N* parameter. You can give the task any name, for example:

```
[Analysis]
AnalysisEngine0=Classification
```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`ObjectClass`**.
`Input`	(Optional) The image track to process. If you do not specify an input track, Media Server processes the first track of the correct type that is produced by the ingest engine.
`Classifier`	The classifier to use for classification. Media Server categorizes objects into the classes in this classifier.
`ClassificationThreshold`	(Optional) The minimum confidence score necessary for Media Server to output a classification result. Any classification result with a confidence score below the threshold is discarded.

   For example:

```
[Classification]
Type=ObjectClass
Classifier=Vechicles
ClassificationThreshold=15
```

   For more information about these parameters, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Classification Results

The following XML shows a single record produced by image classification.

```
<output>
  <record>
    ...
    <trackname>ObjectClass.Result</trackname>
    <ObjectClassResult>
       <id>7fcf8741-9b80-4edd-943e-2d7492467dd4</id>
       <classification>
         <identifier>badger</identifier>
         <confidence>99.26</confidence>
         <metadata>
           <item>
             <key>nocturnal</key>
             <value>true</value>
           </item>
         </metadata>
       </classification>
       <classifier>imagenet</classifier>
       <identifier>badger</identifier>
```

```
      <confidence>99.26</confidence>
    </ObjectClassResult>
  </record>
</output>
```

The record contains the following information:

- The `id` element provides a unique identifier for the classified image. Media Server issues an ID for each image.
- The `classification` element represents a match between the image and an object class in your chosen classifier.
  - The `identifier` element provides the identifier of the object class into which the image was classified.
  - The `confidence` element provides the confidence score for the classification (from 0 to 100).
  - The `metadata` element provides metadata that you have added to the object class in the training database. If there is no metadata associated with the object class, this element is omitted.
- The `classifier` element provides the name of your chosen classifier.
- The `identifier` and `confidence` elements provide the same data as the those in the `classification` element. These elements are deprecated.

Chapter 13: Detect Objects

Object detection uses convolutional neural networks to locate instances of objects that belong to known, pre-defined classes. For example, if you are processing video of a road captured by a CCTV camera, you can configure Media Server to return the locations of all pedestrians, vans, and cars that appear in the video.

Introduction

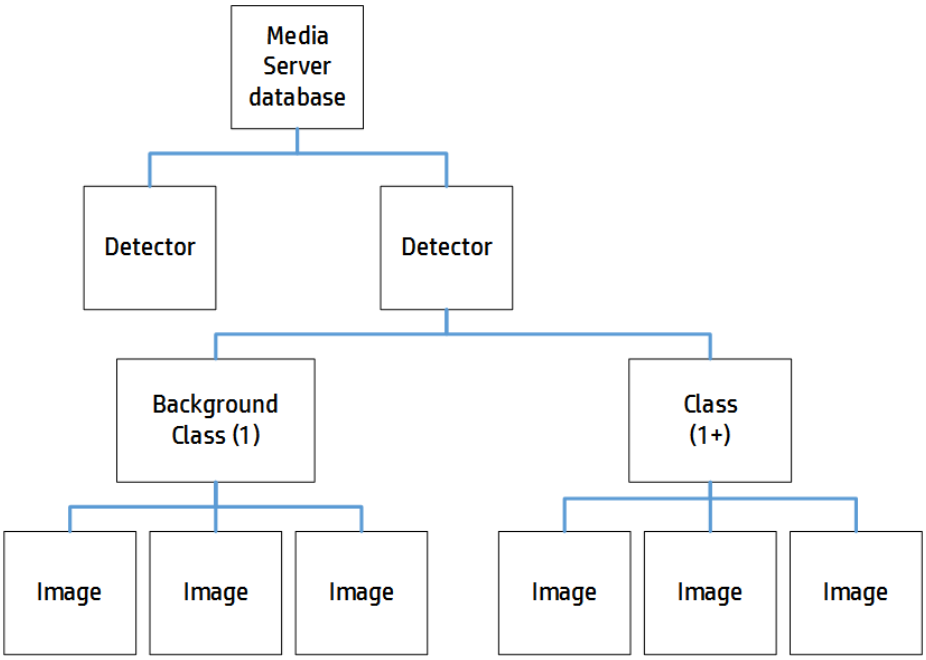
Object detection locates instances of objects that belong to pre-defined classes, such as "car" or "van". This is more general than object recognition, which recognizes specific objects such as a red van that matches a specific model and has the logo of "ABC company" painted on its side.

Object detection differs from image classification because it returns the position for every detected object. Object detection can locate multiple objects within the analyzed image or region, whereas classification returns a single classification result and no position information. For example, using an object detector that has been trained to locate cars and people might return more than one instance of a car and more than one instance of a person in the same image.

Train Media Server to Detect Objects

Before using object detection, you must import a detector that is provided by HPE. The detectors provided by HPE are pre-trained, so you do not need to perform any training. For information about the detectors that are available, see "Pre-Trained Object Detectors" on page 284.

The following diagram shows how Media Server stores information for object detection:



The "Media Server database" represents the Media Server datastore file or a database on an external database server. For information on setting up this database, see "Set up a Database to Store Training Data" on page 29.

The Media Server database can contain any number of detectors. An object detection task uses a single detector and finds instances of objects that are described by classes in that detector. For example, if you are processing video captured by a CCTV camera, you could use the `roadscene` detector which contains classes for finding people, cars, and vans.

A detector contains a background class and a minimum of one foreground class. The background class contains images of uninteresting (background) objects and this class is never returned in a detection result. Only foreground classes are returned in detection results as they represent the categories of objects that you want to detect.

Import a Detector

HPE may provide detectors that you can use with Media Server to run object detection. These detectors are pre-trained, and to use one you only need to import the training data into your Media Server database. For a list of the available pre-trained detectors, see "Pre-Trained Object Detectors" on page 284.

To import a detector

1. Go to https://downloads.autonomy.com/productDownloads.jsp and download the detector to your Media Server machine.
2. Import the training data by running the action `ImportDetector`, for example if you downloaded the file to the folder `pretrained/objectdetection`, in the Media Server installation folder:

```
/action=ImportDetector&detector=roadscene
        &detectorpath=./pretrained/objectdetection/roadscene.dat
```

where,

`detector`	The name to give to the imported detector.
`detectorpath`	The path of the detector data file on disk
`detectordata`	The detector data (you can set this as an alternative to `detectorpath`).

Media Server imports the data. You can now run object detection.

Detect Objects

To detect objects

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=ObjectDetection
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`ObjectDetection`**.
`Input`	(Optional) The image track to process. If you do not specify an input track, Media Server processes the first track of the correct type that is produced by the ingest engine.
`Detector`	The detector to use.
`DetectionThreshold`	(Optional) The minimum confidence score necessary for Media Server to output an object detection result. Any result with a confidence score below the threshold is discarded.

   For example:

   ```
   [ObjectDetection]
   Type=ObjectDetection
   Detector=roadscene
   DetectionThreshold=30
   ```

   For more information about these parameters, refer to the *Media Server Reference*.
4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Object Detection Results

The following XML shows two records produced by object detection.

```
<output>
  <record>
    ...
    <trackname>ObjectDetection.Result</trackname>
    <ObjectDetectionResult>
      <id>d5a249f7-3207-4a41-9a23-0b8ac4aafbf6</id>
      <classification>
        <identifier>van</identifier>
        <confidence>100</confidence>
      </classification>
      <detector>roadscene</detector>
      <region>
        <left>31</left>
        <top>54</top>
        <width>206</width>
        <height>159</height>
      </region>
    </ObjectDetectionResult>
  </record>
  <record>
    ...
    <trackname>ObjectDetection.Result</trackname>
    <ObjectDetectionResult>
      <id>d5a249f7-3207-4a41-9a23-0b8ac4aafbf7</id>
      <classification>
        <identifier>person</identifier>
        <confidence>99.94</confidence>
      </classification>
      <detector>roadscene</detector>
      <region>
        <left>243</left>
        <top>33</top>
        <width>96</width>
        <height>182</height>
      </region>
    </ObjectDetectionResult>
  </record>
</output>
```

Each record contains the following information:

- The `id` element provides a unique identifier for the detected object.
- The `classification` element provides information about the detected object.

    - The `identifier` element provides the identifier of the class that resulted in the object being detected.
    - The `confidence` element provides the confidence score for the detection (from 0 to 100).
- The `detector` element provides the name of the object detector that was used.
- The `region` element provides the location of the detected object in the image or video frame.

Chapter 14: Recognize Objects

Object recognition detects the appearance of specific objects in video; for example, a specific company logo.

Introduction

You can train Media Server to recognize objects, such as logos, that appear in images and video. These objects can be two-dimensional or three-dimensional.

2-D object with 2-D similarity transform

For example, a company logo on a scanned document. Media Server can still recognize the logo when it is subject to 2-D similarity transformations such as rotation and scaling.

2-D object with perspective transform

For example, a photograph or video frame that shows a sign displaying a company logo.

Perspective transformations result from viewing the object from any angle other than directly perpendicular to its plane. This can result in skewing of the image. Perspective transformations often occur in photographs of objects when the camera axis is not directly aligned with the object.

Media Server can only recognize 2-D objects that appear on flat, rigid objects. For example, if you attempt to recognize a logo printed on a flag or item of clothing, recogntion may not be reliable.

3-D object

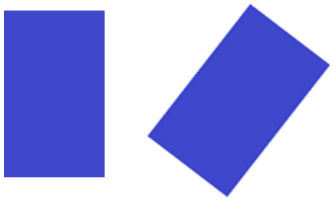
For example, a photograph or video frame that shows product packaging.

If you supply sufficient training images, Media Server can recognize a 3-D object from any angle.

2D Object Recognition

Media Server can recognize 2-D objects in images and video, even when they are subject to similarity or perspective transformation. However, Media Server expects the object to appear on a flat, rigid surface.

The following table provides some examples of transformations that are supported and are not supported.

Transformation	Example	Supported?
Original object		✓
Scaling		✓
Rotation		✓
Perspective distortion or skew (horizontal or vertical only)		✓
Perspective distortion or skew (both horizontal and vertical)		✓

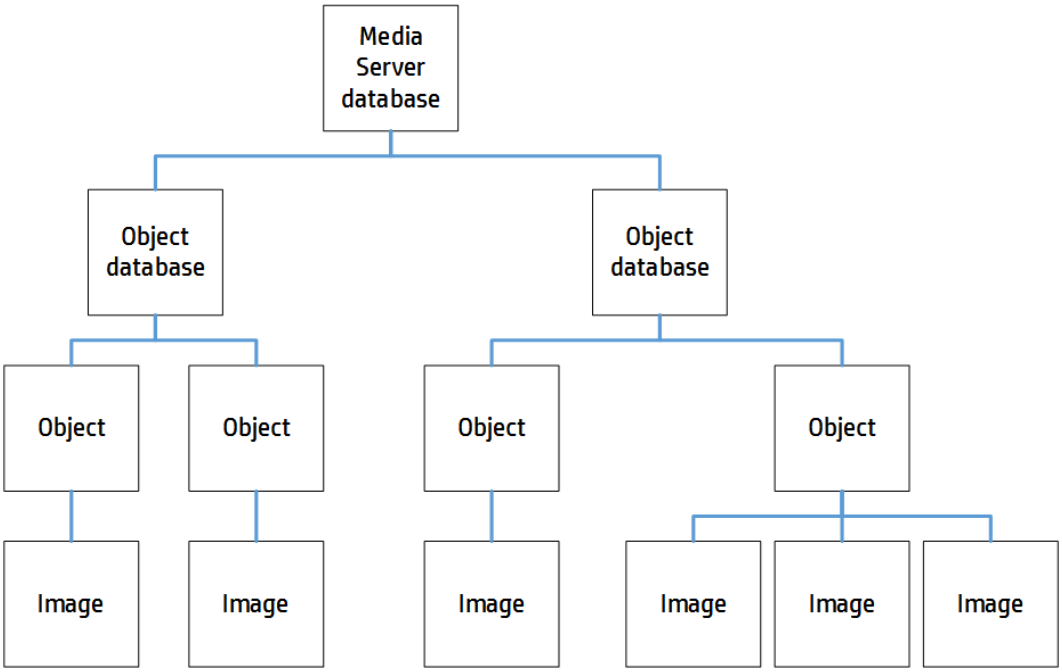
2-D object on a surface that is not flat		✗

Tip: HPE Media Server does not support the recognition of 2-D objects on surfaces that are not flat. However, acceptable accuracy is achievable if you train Media Server using images of the object on the curved surface, rather than images of the object on a flat surface (so that the training is representative of the images you want to analyze).

Train Media Server to Recognize Objects

You must train Media Server by providing images of objects that you want to recognize. When you run object recognition, Media Server uses the training data to recognize objects in your media.

The following diagram shows how Media Server stores information you provide during training.



The "Media Server database" represents the Media Server datastore file or a database on an external database server. For information on setting up this database, see "Set up a Database to Store Training Data" on page 29.

You can organize objects into databases. These object databases are for organizational purposes. For example, you might have a database for company logos, and a database for your company's products. If you want to recognize only one category of objects in your images, you can use only that database.

A database can contain any number of objects, both 2D and 3D. You can associate training options and custom metadata with each object.

To each object you must add a training image (multiple training images for a 3-D object). For information about choosing suitable training images, see "Select Images for Training" below.

Select Images for Training

The following table describes the training requirements for 2-D and 3-D objects:

Object type	Requirements
2-D	One or more training images. Media Server creates a separate model for each training image. For example, if a company has many different variations of its logo, you only need to add one 2-D object to your object database. Add all of the training images for the logo to the same object, even if some variations of the logo do not look like the others.
3-D	Multiple training images depicting the same 3-D object from all angles. Provide images of the object from all angles that you expect it to appear in target images. HPE recommends that the images are taken in one continuous process. One way to obtain training images is by recording a video of the object from all angles and extracting images from the video. As a rough estimate, between 20 and 40 images of an object are usually sufficient to provide accurate detection in a small image. Media Server uses all of the training images to create a single model.

A good training image for an object:

- contains only the object, with as little background as possible. The object should be the dominant feature in the image but there should be at least 16 pixels of background surrounding the object. If any parts of an object touch the image edges, the features corresponding to those parts of the image are likely to be lost.
- includes transparency information, if you intend to recognize the object against many different backgrounds (for example, superimposed on TV footage). Ideally all parts of the image that are not part of the object are transparent (by having their alpha channel set appropriately).

  Note: Only some image formats support transparency information, for example .PNG and .TIFF. Media Server does not support transparency information in .GIF files.

- has not been converted from another file format to JPEG. Converting images to JPEG introduces compression artifacts. If you have JPEG images, avoid making many rounds of edits. Each time that a new version of the image is saved as a JPEG, the image quality degrades.

Create a Database to Contain Objects

To create a database to contain objects, use the following procedure.

To create a database to contain objects

- Use the `CreateObjectDatabase` action, with the `database` parameter:

  `database` The name of the new database (maximum 254 bytes).

  For example:

```
curl http://localhost:14000 -F action=CreateObjectDatabase
                            -F database=CompanyLogos
```

Add an Object to a Database

To add an object to database of objects, you can:

- **Perform training in separate steps**. Create a new (empty) object, then add training images, and then train Media Server, using separate actions. You can also add metadata to the object and configure training options, but these are optional steps. You might want to perform training in this way if you are building a front end application that responds to user input.
- **Perform training in a single action**. You might use this approach when writing a script to train Media Server from a collection of images and metadata.

Add an Object to a Database (Using Separate Steps)

This section describes how to create a new (empty) object, then add training images, and then train Media Server, using separate actions. You can also add metadata to the object, and configure training options, but these are optional steps. You might want to perform training in this way if you are building a front end application that responds to user input.

Alternatively, you can train Media Server to recognize an object by sending a single action. To do this, see "Add an Object to a Database (Using a Single Action)" on page 160.

To add an object to a database (using separate steps)

1. Add an object using the `NewObject` action. Set the following parameters:

   `database` The name of the database to add the object to. The database must already exist.

   `identifier` (Optional) A unique identifier for the object (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.

   For example:

```
curl http://localhost:14000 -F action=NewObject
                            -F database=CompanyLogos
```

   Media Server adds an object to the database and returns the identifier.

2. Add one or more training images to the object using the `AddObjectImages` action. Set the following parameters:

   `database` The name of the database that contains the object.

`identifier`	The identifier for the object, returned by the `NewObject` action.
`imagedata`	The training images to add. Files must be uploaded as multipart/form-data. For more information about sending data to Media Server, see "Send Data by Using a POST Method" on page 67.
`imagelabels`	A comma-separated list of labels. One label is associated with each image. (maximum 254 bytes for each label). The number of labels must match the number of images. If you do not set this parameter, Media Server generates labels automatically.

For example:

```
curl http://localhost:14000 -F action=AddObjectImages
                            -F database=CompanyLogos
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
                            -F imagedata=@logo1.png
```

3. (Optional) Configure the way that Media Server is trained by setting training options for the object. To do this use the `SetObjectTrainingOption` action with the following parameters:

`database`	The name of the database that contains the object.
`identifier`	The identifier for the object, returned by the `NewObject` action.
`key`	The name of the training option that you want to change. For more information about training options, see "Object Training Options" on page 161.
`value`	The new value for the training option.

For example:

```
curl http://localhost:14000 -F action=SetObjectTrainingOption
                            -F database=CompanyLogos
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
                            -F key=useColor
                            -F value=true
```

4. (Optional) Add metadata to the object using the `AddObjectMetadata` action. You can add any number of key-value pairs. Set the following parameters:

`database`	The name of the database that contains the object.
`identifier`	The identifier for the object, returned by the `NewObject` action.
`key`	The key to add (maximum 254 bytes).
`value`	The value to add (maximum 254 bytes).

For example:

```
curl http://localhost:14000 -F action=AddObjectMetadata
                            -F database=CompanyLogos
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
```

```
                                -F key=CompanyName
                                -F value=HewlettPackard
```

5. Complete the training for the object using the `BuildObject` action. Set the following parameters:

`database`	The name of the database that contains the object.
`identifier`	The identifier for the object, returned by the `NewObject` action.

   For example:

```
curl http://localhost:14000 -F action=BuildObject
                            -F database=CompanyLogos
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
```

Add an Object to a Database (Using a Single Action)

You can train Media Server to recognize an object by sending a single action (`TrainObject`).

Running this action is equivalent to running the following actions in the following order:

- `NewObject`
- `AddObjectImages`
- `SetObjectTrainingOption` (optional)
- `AddObjectMetadata` (optional)
- `BuildObject`

The `TrainObject` action is atomic, so that any interruption to the server does not leave the database in an inconsistent state.

Alternatively, you can train Media Server by sending these actions individually. You might want to do this if you are building a front end application that trains Media Server in response to user input. For more information about how to do this, see "Add an Object to a Database (Using Separate Steps)" on page 158.

To add an object to a database (using a single action)

- Add an object using the `TrainObject` action. Set the following parameters:

`database`	The name of the database to add the object to. The database must already exist.
`identifier`	(Optional) A unique identifier for the object (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.
`imagedata`	The training images to add. Files must be uploaded as multipart/form-data. For more information about sending data to Media Server, see "Send Data by Using a POST Method" on page 67.
`imagelabels`	(Optional) A comma-separated list of labels. One label is associated with each image. (maximum 254 bytes for each label). The number of labels must match the number of images. If you do not set this parameter, Media Server generates labels automatically.
`metadata`	(Optional) A comma-separated list of metadata key-value pairs to add to the object. Separate keys from values using a colon (:). To include a comma or colon in a key name or value, you must enclose the key name or value in quotation marks (") and escape any quotation marks that occur within the string with a backslash (\).
`trainingoptions`	(Optional) A comma-separated list of training options to apply to the object. Separate training options from their values using a colon (:).

For example:

```
curl http://localhost:14000 -F action=TrainObject
                            -F database=CompanyLogos
                            -F imagedata=@logo.png
                            -F metadata=CompanyName:HPE,"another:value":"1,000"
                            -F trainingoptions=useColor:true,3D:false
```

Media Server adds the object to the database and returns the identifier.

Object Training Options

After you create a new object in an object database, you can set training options for the object. Training options configure how Media Server uses the training data that you supply to create a model of the object.

You can set the following training options:

Training Option	Description	Acceptable values	Default value
`3D`	Specifies whether the object is a three-dimensional object.	true, false	false
`boundaryFeatures`	Specifies whether there are important features at the	true, false	false

	edges of the training images. For example if you supply a training image of a rectangular flag, and the edge of the image represents the edge of the flag, set this option to `true`.		
`useColor`	Specifies whether Media Server adds color information from the training images into the models that it creates for recognition. For example, if you are recognizing company logos but one company often prints its logo in different colors, set this option to `false`.	true, false	false

For an example that shows how to add a new object and set training options, see "Add an Object to a Database" on page 158.

List the Objects in a Database

To list the objects that you have added to a database, and check whether training was successful, use the following procedure.

To list the objects in a database

1. (Optional) First list the databases that have been created to store objects. Use the action `ListObjectDatabases`:

   ```
   http://localhost:14000/action=ListObjectDatabases
   ```

   Media Server returns a list of databases that you have created.

2. List the objects that exist in one of the databases. Use the action `ListObjects`, for example:

   ```
   http://localhost:14000/action=ListObjects&database=logos
                                            &metadata=true
                                            &trainingoptions=true
                                            &imagestatus=true
   ```

   Media Server returns a list of objects in the specified database, and the number of training images associated with each object.

   If you set the action parameter `metadata` to **`true`**, Media Server returns the metadata you have added to the object.

   If you set the action parameter `trainingoptions` to **`true`**, Media Server returns the training options you have set for the object.

   If you set the action parameter `imagestatus` to **`true`**, Media Server returns the status of each training image associated with each object.

   - A status of `trained` indicates that training was successful.
   - If images have a status of `untrained`, run training for the object using the action `BuildObject`.
   - If images have a status of `failed`, Media Server could not use the image for training. Remove the failed images using the action `RemoveObjectImages`.

Update or Remove Objects and Databases

To update or remove an object use the following actions:

- To add additional training images to an object, use the action `AddObjectImages`. Media Server does not use the new images for object detection until you run the action `BuildObject`. To confirm that training was successful, use the action `ListObjects` (see "List the Objects in a Database" on the previous page).
- To remove a training image from an object, use the action `RemoveObjectImages`.
- To modify the training options for an object, use the actions `SetObjectTrainingOption` and `UnsetObjectTrainingOption`.
- To change the label of an image that you added to an object, use the action `RelabelObjectImage`.
- To move an image of an object from one object to another, for example if you add an image to the wrong object, use the action `MoveObjectImage`.
- To add, remove, or update custom metadata for an object, use the actions `AddObjectMetadata`, `RemoveObjectMetadata`, and `UpdateObjectMetadata`.
- To change the identifier of an object you added to an object database, use the action `RenameObject`.
- To remove an object from a database and discard all associated images and metadata, use the action `RemoveObject`.

To update or remove object databases, use the following actions:

- To rename an object database, use the action `RenameObjectDatabase`.
- To delete an object database and all of the information that it contains, use the action `RemoveObjectDatabase`.

For more information about the actions that you can use to train Media Server, refer to the *Media Server Reference*.

Synchronize with the Latest Training

Training Media Server updates the training database. However, the training is not used in analysis operations until Media Server has synchronized with the database. Synchronization loads the data from the database into memory and makes it ready for use. Synchronization is necessary because several Media Servers can share a single database, and you can use any Media Server to perform training.

The default configuration ensures that analysis uses the latest training data, but you can customize Media Server for your use case.

Automatic Synchronization

The default configuration ensures that analysis uses the latest training data, no matter how often you train Media Server and how much training data you add to the database.

Before it starts a `process` action that requires training data, Media Server synchronizes with the training database and waits for the synchronization to complete. This is because the `SyncDatabase` parameter - available for face recognition, object recognition, and image classification tasks - has a default value of `TRUE`.

Media Server also synchronizes with the database at regular intervals (by default, every second). You can configure the interval between each synchronization by setting the `SyncInterval` parameter in the `[Database]` section of the configuration file. This is important for processing video streams, because when you process a stream Media Server could continue running the same action for days or weeks. Automatic synchronization ensures that Media Server regularly loads the latest training without you having to intervene.

Manual Synchronization

In some cases, you might want to control when Media Server synchronizes with the training database:

- In a production environment, you might modify the training only at known times. In this case you might prefer to disable automatic synchronization, because forcing Media Server to query the database before it starts processing each request can reduce the processing speed.
- You might want to disable automatic synchronization to ensure that a batch of `process` requests all use the same training. If you are processing discrete files, you might perform training in bulk and then process batches of files.

To change the frequency at which Media Server automatically synchronizes with the database, or disable scheduled synchronization completely, set the parameter `SyncInterval` in the `[Database]` section of the configuration file.

To prevent Media Server synchronizing with the database before it starts a `process` action, set `SyncDatabase=False` in the relevant analysis tasks. With this setting, be aware that `process` requests could use outdated training. This is especially likely if you add large amounts of training data to the database and then immediately start a `process` action, because Media Server might not have finished training before it starts processing.

If you disable automatic synchronization but want to ensure that Media Server has synchronized with the latest training, run the `SyncFaces`, `SyncObjects`, and `SyncClassifiers` actions before you send the `process` action. These actions force Media Server to synchronize with the database. After they complete, you can be sure that Media Server is using the latest training data.

Reduce Memory Use

With automatic synchronization, described above, Media Server loads all training data into memory. If you have an extremely large training database that contains many face databases, object databases, and classifiers, this can consume a significant amount of memory.

You can remove all face databases, object databases, or classifiers from memory by setting `SyncInterval=0` (to disable the automatic synchronization) and running the action `UnsyncFaces`, `UnsyncObjects`, or `UnsyncClassifiers`.

You can then load the training you need using one of the following methods:

- In the `[`*`TaskName`*`]` section for your analysis tasks, set `SyncDatabase=TRUE`. Media Server will load the training it needs to complete the tasks.
- In the `[`*`TaskName`*`]` section for your analysis task, set `SyncDatabase=FALSE`. Then, manually load the training data you need by running the actions `SyncFaces`, `SyncObjects`, and `SyncClassifiers`. You can add the `database` parameter to the `SyncFaces` and `SyncObjects` actions to choose which database to load into memory.

IDOL Admin

You can train Media Server using IDOL Admin. To open the IDOL Admin interface, send the following action to Media Server's ACI port:

```
http://localhost:14000/action=admin
```

For more information about using IDOL Admin refer to the *IDOL Admin User Guide*.

Recognize Objects

To recognize objects, configure an object analysis task by following these steps.

To recognize objects

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=Object
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`object`**.
`Input`	(Optional) The image track to process.
`Database`	(Optional) The database to use for recognizing objects. By default, Media Server searches for objects from all object databases. Database names are case-sensitive.
`ColorAnalysis`	(Optional) A Boolean value that specifies whether to check the color of detected objects in order to reduce false identification. Set this parameter if the objects are primarily distinguished by color; for example, some flags differ from each other by color only.
`ObjectEnvironment`	(Optional) Some objects, such as logos, might be partially or completely transparent, meaning that their appearance changes depending on the background on which they are superimposed. In such cases, set this parameter to specify the type of background in target images. **Note:** For the `ObjectEnvironment` parameter to have an effect, the image of the object used for training the database must contain transparent pixels.
`Occlusion`	(Optional) By default, Media Server assumes that an object might be partially hidden in target images, for example by overlapping objects. If

	the object is never obscured in target images, set this parameter to **FALSE** to reduce false positives.
`Region`	(Optional) Search for objects in a region of the image, instead of the entire image.

For example:

```
[Object]
Type=Object
Database=CompanyLogos
```

For more details about the parameters that you can use to customize object recognition, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Object Recognition Results

The following XML shows a single record produced by object recognition.

```
<output>
  <record>
    ...
    <trackname>object.Result</trackname>
    <ObjectResult>
      <id>ed1f3af7-9f00-434f-8bc4-a328ff4c67fc</id>
      <identity>
        <identifier>HPE</identifier>
        <database>logos</database>
        <imagelabel>752fdee3b5c478f7314eca75365c4094</imagelabel>
        <confidence>100</confidence>
        <metadata>
          <item>
            <key>CompanyName</key>
            <value>HPE</value>
          </item>
        </metadata>
      </identity>
      <boundary>
        <point>
          <x>106</x>
          <y>100</y>
        </point>
        <point>
          <x>271</x>
          <y>101</y>
        </point>
        <point>
```

```
          <x>272</x>
          <y>183</y>
        </point>
        <point>
          <x>107</x>
          <y>183</y>
        </point>
      </boundary>
    </ObjectResult>
  </record>
</output>
```

The record contains the following information:

- The `id` element provides a unique identifier for the recognized object. Media Server issues an ID for each appearance of an object. If you are recognizing objects in video and consecutive frames show the same object in a near-identical location, all records related to that appearance will have the same ID.

  For example, if an object appears in the same location for a hundred consecutive video frames, the engine uses the same ID for each record in the data track and the single record in the result track. The record in the result track will have a timestamp that covers all of the frames.

  If the object disappears and then reappears, the engine considers this as a new detection and produces a new ID and a new record in the result track.
- The `identity` element represents a match between the ingested media and an object in your training database.
  - The `identifier` element provides the identifier of the object that was detected in the ingested media.
  - The `database` element provides the name of the database in which the object exists.
  - The `imagelabel` element provides the label of the image that was the best match to the analyzed media (many training images can be associated with a single object in the database). This element is present only for 2D objects. With 3D objects all of the training images are used to define a single representation of the object.
  - The `confidence` element provides the confidence score for the match (from 0 to 100).
  - The `metadata` element provides metadata that you associated with the object when you trained Media Server. If there is no metadata in the training database, this element is omitted.
- The `boundary` element provides the position of the object in the ingested media, as a set of points that form a polygon which surrounds the object.

Optimize Object Recognition Performance

The quality of the image or video that you send to Media Server can have a significant effect on the performance of object recognition.

Consider the following expectations for input images and video:

- The size of the object within the image or video frame is important. Object recognition works reliably (depending on other factors) if the object occupies a minimum area of 100x100 pixels. Some objects can be recognized down to 50x50 pixels, but Media Server does not usually recognize any object smaller than this. The size of the image or video is less important than the size of the object; however a large image might contain a large amount of clutter, which means that object recognition might take longer.
- Objects should not appear blurry.
- Objects can be recognized if they are partially obscured (set the `Occlusion` configuration parameter), but this might reduce accuracy.
- Object recognition performs best when the image or video has even illumination, because dim lighting, dark shadows, or bright highlights can reduce accuracy.
- The image or video should not be affected by significant compression artifacts that can affect some formats (such as highly compressed JPEG images). HPE recommends that you do not save your image files as JPEG images with high levels of compression or transcode video that has been captured from a camera. If you are using a digital video recorder (DVR) to record the footage from a camera and are then sending the video to Media Server, ensure the DVR is saving the video at full resolution and is not reducing the bit rate.

Chapter 15: Analyze Audio

Media Server can use an IDOL Speech Server to analyze audio.

Introduction

Media Server can perform the following types of analysis on speech:

- **Language identification** identifies the language of the speech in the audio.
- **Speech-to-text** extracts speech and converts it into text.
- **Speaker identification** identifies the people who speak in the video.
- **Audio matching** identifies when known audio clips appear in the ingested media. You can use audio matching to help identify copyright infringement if copyrighted music is played or detect specific advertisements in ingested video.

When the audio or video source contains narration or dialogue, running speech analysis and indexing the resulting metadata into IDOL Server means that IDOL can:

- search all of the video that you have processed to find clips where a speaker talks about a specific topic.
- provide parametric search to filter the results by speaker.
- categorize video clips.
- cluster together video clips that contain related concepts.

To analyze speech, you must have an IDOL Speech Server. For more information about using IDOL Speech Server, refer to the IDOL Speech Server documentation.

Identify the Language of Speech

To identify the language of speech

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*N* parameter. You can give the task any name, for example:

```
[Analysis]
AnalysisEngine0=SpeechLanguageId
```

3. Create a new section to contain the settings for the task, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`LanguageID`**.
`Input`	(Optional) The audio track to process. If you do not specify an input track, Media Server processes the first audio track produced by the ingest engine.
`LanguageIdServers`	The host name and ACI port of an IDOL Speech Server. Separate the host name and port with a colon (for example, `speechserver:15000`). You can specify multiple IDOL Speech Servers by using a comma-separated list. Media Server can connect to only one IDOL Speech Server at a time, but you can provide multiple servers for failover. **Tip:** You can specify a default IDOL Speech Server to use for all language identification tasks by setting the `LanguageIdServers` parameter in the `[Resources]` section of the Media Server configuration file.
`LangList`	(Optional) The list of languages to consider when running language identification. If you know which languages are likely to be present in the media, HPE recommends setting this parameter because restricting the possible languages can increase accuracy and improve performance.
`CumulativeMode`	(Optional, default `false`) A Boolean that specifies whether to run analysis in cumulative mode. If you expect the audio to contain only one language or you want to identify the primary language that is spoken in the audio, set this parameter to **`true`**. Media Server outputs results to the result track after analyzing each audio segment but every result is based on analysis of the current segment and all of the previous segments. This mode is not suitable for analyzing continuous streams. If you set this parameter to **`false`**, Media Server runs analysis in segmented mode. The audio is segmented into fixed-size segments and Media Server returns a language identification result for each segment. Media Server does not consider previous segments when running analysis. You can use this mode to determine if there are multiple languages present in the audio, but this mode is not intended to identify the boundary points where the language changes.
`SegmentSize`	(Optional, default `15`) The amount of audio to analyze as a single segment, in seconds.

For example:

```
[SpeechLanguageId]
Type=LanguageID
LanguageIDServers=speechserver:15000
CumulativeMode=True
SegmentSize=30
```

For more information about the parameters that you can use to configure this task, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Transcribe Speech

To run speech-to-text

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngineN` parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=TranscribeSpeech
   ```

3. Create a new section to contain the settings for the task and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`speechtotext`**.
`Input`	(Optional) The audio track to analyze. If you do not specify an input track, Media Server processes the first track of the correct type produced by the ingest engine.
`SpeechToTextServers`	The host name and ACI port of an IDOL Speech Server. Separate the host name and port with a colon (for example, `speechserver:15000`). You can specify multiple IDOL Speech Servers by using a comma-separated list. Media Server can connect to only one IDOL Speech Server at a time, but you can provide multiple servers for failover. **Tip:** You can specify a default IDOL Speech Server to use for all speech-to-text tasks by setting the `SpeechToTextServers` parameter in the `[Resources]` section of the Media Server configuration file.
`Language`	The language pack to use. For a list of available language packs, refer to the *IDOL Speech Server Administration Guide*.
`Mode`	The mode for speech-to-text transcription. Media Server supports the following modes: • `Fixed`

	• `Relative` These modes are described in the *IDOL Speech Server Reference*.
`ModeValue`	The mode value for speech-to-text transcription. For more information about this parameter, refer to the *IDOL Speech Server Reference*.
`FilterMusic`	(Optional) Specifies whether to ignore speech-to-text results for audio segments that Speech Server identifies as music or noise. To filter these results from the output, set this parameter to `true`.
`SampleFrequency`	(Optional) The sample frequency of the audio to send to the IDOL Speech Server for analysis, in samples per second (Hz). IDOL Speech Server language packs are dependent on the audio sample rate, and accept audio at either 8000Hz or 16000Hz.

For example:

```
[TranscribeSpeech]
Type=speechtotext
SpeechToTextServers=speechserver:13000
Language=ENUK
Mode=relative
ModeValue=0.8
FilterMusic=TRUE
```

For more information about the parameters that you can use to configure speech-to-text, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Identify Speakers

Note: Before IDOL Speech Server can identify speakers, you must train the Speech Server. Without training, Speech Server can divide the audio into different speakers and identify the gender of each speaker. For information about training IDOL Speech Server, refer to the *IDOL Speech Server Administration Guide*.

To identify speakers in video

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*N* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=IDspeakers
   ```

3. Create a new section to contain the settings for the task, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`speakerid`**.
`Input`	(Optional) The audio track to process. If you do not specify an input track, Media Server processes the first track of the correct type produced by the ingest engine.
`SpeakerIDServers`	The host name and ACI port of an IDOL Speech Server. Separate the host name and port with a colon (for example, `speechserver:13000`). You can specify multiple IDOL Speech Servers by using a comma-separated list. Media Server can connect to only one IDOL Speech Server at a time, but you can provide multiple servers for failover. **Tip:** You can specify a default IDOL Speech Server to use for all speaker identification tasks by setting the `SpeakerIdServers` parameter in the `[Resources]` section of the Media Server configuration file.
`TemplateSet`	(Optional) The path to the audio template set (.ats) file to use for speaker identification. You must create this file. Specify the path relative to the directory defined by the `SpeakerIDDir` parameter in the IDOL Speech Server configuration file. If you do not set this parameter Speech Server cannot identify speakers, but can divide the audio into different speakers and detect the gender of each speaker. **Tip:** If you are sending audio to IDOL Speech Server version 10.x, set the parameters `AstPath` and `UseLegacyAction=TRUE` instead.
`SampleFrequency`	(Optional) The sample frequency of the audio to send to the IDOL Speech Server for analysis, in samples per second (Hz). IDOL Speech Server accepts audio at either 8000Hz or 16000Hz.

For example:

```
[IDspeakers]
Type=speakerid
SpeakerIDServers=speechserver:13000
TemplateSet=speakers.ats
```

For more information about the parameters that you can use to configure this task, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Recognize Audio Clips

Note: Before IDOL Speech Server can recognize audio clips, you must train the Speech Server. For information about training IDOL Speech Server, refer to the *IDOL Speech Server Administration Guide*.

To recognize audio clips

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*N* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=AudioMatch
   ```

3. Create a new section to contain the settings for the task and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`AudioMatch`**.
`Input`	(Optional) The audio track to analyze. If you do not specify an input track, Media Server processes the first track of the correct type produced by the ingest engine.
`Pack`	The name of the audio fingerprint database to use for recognizing audio clips. You must train Speech Server before configuring Media Server.
`AudioMatchServers`	The host name and ACI port of the Speech Server(s) to use for audio matching. Separate the host name and port with a colon (for example, `speech:15000`). You can specify multiple IDOL Speech Servers by using a comma-separated list. Media Server can connect to only one IDOL Speech Server at a time, but you can provide multiple servers for failover. **Tip:** You can specify a default IDOL Speech Server to use for all audio match tasks by setting the `AudioMatchServers` parameter in the `[Resources]` section of the Media Server configuration file.
`SampleFrequency`	(Optional) The sample frequency of the audio to send to the IDOL Speech Server for analysis, in samples per second (Hz).

   For example:

   ```
   [AudioMatch]
   Type=AudioMatch
   Pack=Music
   AudioMatchServers=speech:15000
   SampleFrequency=16000
   ```

   For more information about the parameters that you can use to configure audio matching, refer to the *Media Server Reference*.
4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 16: Segment Video into News Stories

News segmentation analyzes news broadcasts, identifies the times at which new stories begin and end, and extracts the key concepts from each story.

Introduction

News segmentation analyzes news broadcasts, identifies the times at which stories start and finish, and extracts the key concepts from each story.

News segmentation classifies the ingested video into segments that can be:

- A **story** - a video segment that contains a consistent set of concepts.
- A **short story** - a video segment that contains a consistent set of concepts, but the total length is too short to be considered a complete story. For example, a news presenter might briefly describe the news headlines at the end of the hour, so each topic is likely to be covered for only 20 or 30 seconds.
- **No topic** - a video segment that does not contain a consistent set of concepts.

The news segmentation task is intended to be used on speech that has been transcribed from the audio track of a news broadcast, and segmented into sentences by a text segmentation task.

News segmentation enables Media Server to output documents that correspond to stories in a news broadcast. When you configure Media Server to output documents, use bounded event mode and use the news segmentation result track as the event track. For more information about configuring Media Server to output documents, see "Output Data" on page 240.

Prerequisites

To perform news segmentation you must have an IDOL Server (Content component) that contains recent examples of news stories that are relevant to the news channel that you are analyzing. The Content component should contain no other data, because Media Server uses all documents, across all databases, to aid classification.

Configure News Segmentation

To configure news segmentation , follow these steps.

To configure news segmentation

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add three analysis tasks. The first should run speech-to-text on an audio track of the ingested video. The second should run text segmentation to group the words extracted by speech-to-text into sentences. The third is the news segmentation task. For example:

   ```
   [Analysis]
   AnalysisEngine0=SpeechToText
   AnalysisEngine1=TextSegmentation
   AnalysisEngine2=NewsSegmentation
   ```

3. Create a new configuration section and configure the speech-to-text task. For information about how to set up speech-to-text, see "Analyze Audio" on page 169. For example:

   ```
   [SpeechToText]
   Type=SpeechToText
   Language=ENUS
   Mode=relative
   ModeValue=0.90
   FilterMusic=true
   ```

4. Create a new configuration section and configure the text segmentation task. For example:

   ```
   [TextSegmentation]
   Type=TextSegmentation
   Input=SpeechToText.Result
   ```

5. Create a new configuration section to contain the news segmentation task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`NewsSegmentation`**.
`Input`	(Optional) The track to process. HPE recommends that you transcribe speech from the audio track of the news broadcast and then group the extracted words into sentences using a text segmentation task. Set this parameter to the result track of the text segmentation task.
`IdolHost`	The host name or IP address of the IDOL Server (Content component) to use for news segmentation.
`IdolPort`	The ACI port of the IDOL Server (Content component) to use for

	news segmentation.
MaxStoryDuration	The maximum duration of a video segment that can be classified as a story. If a story exceeds this duration, Media Server begins a new story regardless of whether the concepts in the video have changed.

For example:

```
[NewsSegmentation]
Type=NewsSegmentation
Input=TextSegmentation.Result
IdolHost=localhost
IdolPort=9000
MaxStoryDuration=30minutes
```

For more information about the parameters that you can use to configure news segmentation, refer to the *Media Server Reference*.

6. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example Configuration

The following is an example configuration that runs news segmentation on ingested video. This configuration runs segmentation using an IDOL Server that is on the same machine as Media Server. It also includes an output task to write the results of segmentation to files on disk.

```
[Ingest]
IngestEngine=libav

[libav]
Type=libav

[Analysis]
AnalysisEngine0=SpeechToText
AnalysisEngine1=TextSegmentation
AnalysisEngine2=NewsSegmentation

[SpeechToText]
Type=SpeechToText
Language=ENUS
Mode=relative
ModeValue=0.90
FilterMusic=true

[TextSegmentation]
Type=TextSegmentation
Input=SpeechToText.Result

[NewsSegmentation]
```

```
Type=NewsSegmentation
Input=TextSegmentation.Result
IdolHost=localhost
IdolPort=9000

[Output]
OutputEngine0=XML

[XML]
Type=xml
Input=NewsSegmentation.Result,SpeechToText.Result
Mode=bounded
EventTrack=NewsSegmentation.Result
XMLOutputPath=./output/%token%/Story_%segment.type%_%segment.sequence%.html
```

News Segmentation Results

The following XML shows a single record produced by news segmentation.

```
<NewsSegmentationResult>
  <id>af4475f5-beb2-401e-bf73-fb494f63af27</id>
  <storyText>business secretary says the government will consider co investing
             with a Buy on commercial terms <SIL> ...</storyText>
  <score>0</score>
  <terms>
    <term>
      <text>Buy</text>
      <weight>170</weight>
    </term>
    <term>
      <text>business secretary says</text>
      <weight>120</weight>
    </term>
    <term>
      <text>consider co investing</text>
      <weight>110</weight>
    </term>
    <term>
      <text>commercial terms</text>
      <weight>90</weight>
    </term>
    ...
  </terms>
  <type>Short story</type>
</NewsSegmentationResult>
```

The record contains the following elements:

- `storyText` contains the text extracted from the video. The text is extracted from the audio by speech-to-text. Words are then combined into segments by the text segmentation task. News Segmentation analyzes the segments and combines one or more segments into a result that represents a story, short story, or video with no topic.
- `score` is an integer from 0 to 100 that indicates the consistency between the terms in each segment that makes up the `storyText`.
- each `terms/term/text` element contains a key term extracted from the text.
- each `terms/term/weight` element contains the weight of the associated term. You might want to use this weight when building a term cloud. The weights are relative to the other terms in the story, and cannot be compared across stories.
- the `type` element specifies whether the news segmentation task has classified the segment as a `Story`, `Short Story`, or a segment with `No topic`.

Chapter 17: Analyze Vehicles

Media Server can detect and analyze vehicles in video. These features are suitable for many different applications. You can use Media Server to monitor car parks, provide a drive-off deterrent at petrol filling stations, and detect stolen vehicles.

Introduction

Media Server can:

- **Detect and read the number plates of vehicles in the scene**. Number plate recognition has many applications; you can detect stolen and uninsured vehicles, and monitor the length of stay for vehicles in car parks.
- **Identify the manufacturer and model of a vehicle identified during number plate recognition**. By comparing the model detected by Media Server to a database, you can use this feature to identify vehicles that have false number plates.
- **Identify the color of a vehicle identified during vehicle model detection**.

Requirements for ANPR

For reliable number plate detection and recognition, ensure that your system meets the following requirements.

Camera	HPE recommends a separate camera for each lane of traffic. If you use a single camera for several lanes of traffic, high-definition recording is required (1080p or better). Manually focus the camera on the middle of the region where number plates are read, and use a fast shutter speed to avoid blurring.
Image contrast	The contrast must be sufficient for the number plates to be human-readable. If you are reading retroreflective number plates, the best results are obtained with infra-red (IR) illumination.
Number plate size	The number plates must be human-readable and characters in the video must not be less than 10 pixels high.

Number plate rotation	Position the camera above the traffic, so that number plates appear to be horizontal. The closer the plates are to horizontal, the better the performance.
Video frame rate	Media Server improves accuracy by reading number plates across multiple frames. For moving traffic Media Server requires 25 or 30 frames per second. Do not reduce the frame rate of the captured video.

Detect and Read Number Plates

Media Server can detect and read number plates that appear in video.

To detect and read number plates in video

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=ANPR
   ```

3. Create a new section in the configuration file to contain the task settings and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`numberplate`**.
`Input`	(Optional) The image track to process. If you do not specify an input track, Media Server processes the first track of the correct type that is produced by the ingest engine.
`Location`	The location in which you are reading number plates. There are significant differences in the number plates used by countries and states, so this parameter is necessary to read the plates successfully.
`Region`	(Optional) The region of interest (ROI) to monitor for number plates (left, top, width, height). If you do not specify a region, Media Server detects and reads number plates anywhere in the scene.
`RegionUnit`	(Optional) The units used to specify the position and size of the region of interest (**`pixel`** or **`percent`**).
`Sensitivity`	(Optional) The confidence level required to detect a number plate.
`Integration`	(Optional) The method to use to combine the number plates read from successive frames into a single result (**`none`**, **`moving`**, or **`static`**).
`MinValidScore`	(Optional) The average character score required for a number plate to be recognized.

For example:

```
[ANPR]
Type=numberplate
Location=uk
RegionUnit=percent
Region=20,10,60,90
Sensitivity=14
Integration=moving
```

For more information about the parameters that customize number plate recognition, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Train Media Server to Recognize Vehicles

Before using vehicle model identification, you must train Media Server by providing images of the vehicle models that you want to recognize. Vehicle identification performs best when vehicles are moving towards the camera, when the front of the vehicle is visible in the video.

Vehicles must be added to an object database as two-dimensional objects.

Obtain Training Images

You can obtain training images by running vehicle model identification on a sample video. Output the result images to the image encoder. For example, you could use the following configuration:

```
[Ingest]
IngestEngine=libav

[libav]
Type=libav

[Analysis]
AnalysisEngine0=ANPR
AnalysisEngine1=VehicleModel

[ANPR]
Type=numberplate
Location=uk

[VehicleModel]
Type=vehiclemodel
Input=ANPR.DataWithSource

[Transform]
TransformEngine0=Crop
```

```
[Crop]
Type=Crop
Input=VehicleModel.ResultWithSource

[Encoding]
EncodingEngine0=VehicleImages

[VehicleImages]
Type=ImageEncoder
ImageInput=Crop.Output
OutputPath=./training/image-%segment.sequence%.jpg
```

The vehicle model analysis engine requires as input either the `DataWithSource` or `ResultWithSource` track from a number plate analysis task.

Usually, the vehicle model analysis task would include the `Database` parameter, which specifies the database to use for recognizing vehicles. If you are obtaining training images, you can run vehicle identification without setting this parameter.

Before you output images using the image encoder, configure a `Crop` transform task so that the images are cropped to the region identified by the vehicle model analysis engine.

Train Media Server

Vehicle identification is trained in the same way as object recognition (see "Train Media Server to Recognize Objects" on page 156). Vehicles must be trained as two-dimensional objects.

Tip: The following procedure explains how to train Media Server by sending actions to Media Server's ACI port. However, you can also train Media Server using IDOL Admin. For information about opening IDOL Admin, see "Access IDOL Admin" on page 65. For information about using IDOL Admin, refer to the *IDOL Admin User Guide*.

To train Media Server to recognize vehicles

1. Create a new object database. Use the `CreateObjectDatabase` action, with the `database` parameter:

`database`	The name of the new database (maximum 254 bytes).

   For example:

   ```
   curl http://localhost:14000 -F action=CreateObjectDatabase
                               -F database=vehicles
   ```

2. Add each vehicle to the database using the `TrainObject` action. Use the following parameters:

`database`	The name of the database to add the object to. The database must already exist.
`identifier`	A unique identifier for the object (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.

`imagedata`	The training images to add. Files must be uploaded as multipart/form-data. For more information about sending data to Media Server, see "Send Data by Using a POST Method" on page 67.
`imagelabels`	A comma-separated list of labels to associate with the images that you are adding (maximum 254 bytes for each label). The number of labels must match the number of images provided. If you do not set this parameter, Media Server generates labels automatically.
`trainingoptions`	A comma-separated list of training options to apply to the object. Separate training options from their values using a colon (:). For information about the training options that you can set, see "Object Training Options" on page 161.

For example:

```
curl http://localhost:14000 -F action=TrainObject
                            -F database=vehicles
                            -F identifier=ford_focus
                            -F imagedata=@image-1.jpg,image-27.jpg,image-33.jpg
                            -F imagelabels=2013focus,2005focus,2002focus
                            -F trainingoptions=useColor:true,3D:false
```

3. Verify that the training was successful.
   - The `TrainObject` action is asynchronous. To obtain the response to these actions, use the `QueueInfo` action. For example:

     ```
     curl
     "http://localhost:14000/a=QueueInfo&queuename=trainobject&queueaction=getstat
     us"
     ```

   - To check the status of the vehicles database, and verify that training was successful, use the `ListObjects` action. For example:

     ```
     curl "http://localhost:14000/a=ListObjects&database=vehicles"
     ```

Identify Vehicle Models

To identify vehicles in video, use the following procedure.

To identify vehicles

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*N* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=ANPR
   AnalysisEngine1=VehicleModel
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`vehiclemodel`**.
`Input`	The track to process. This track must be the `DataWithSource` or `ResultWithSource` track from a number plate analysis task.
`Database`	The name of the object database to use to identify vehicles. For information about creating this database, see "Train Media Server to Recognize Vehicles" on page 182.
`Perspective`	(Optional) If the video frames show the vehicle from a different perspective to that used in the training images, set this parameter to `TRUE` so that Media Server compensates accordingly.
`ColorRegion`	(Optional) The region to output to the `ColorRegionWithSource` output track, so that you can configure an analysis task to identify the color of the vehicle. If you don't specify a region, Media Server uses a default region.

   For example:

```
[VehicleModel]
Type=vehiclemodel
Input=ANPR.DataWithSource
Database=vehicles
Perspective=FALSE
```

   For more information about the parameters that you can use, including the values that they accept, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Identify Vehicle Colors

Media Server can detect the color of vehicles identified by vehicle model detection.

To detect the color of vehicles

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

```
[Analysis]
AnalysisEngine0=ANPR
AnalysisEngine1=VehicleModel
AnalysisEngine2=VehicleColor
```

3. Create a new section in the configuration file to contain the task settings and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`ColorCluster`**.
`Input`	The image track to process. This track must be the `ColorRegionWithSource` output track from a vehicle model analysis task. This track contains a region identified by the vehicle model task, which should contain a sample of the vehicle's color. You can customize the selection of this region by setting the parameter `ColorRegion` in your vehicle model analysis task.
`RestrictToInputRegion`	A Boolean value that specifies whether to analyze a region of the input image or video frame that is specified in the input record, instead of the entire image. Set this parameter to **`TRUE`**, because you want the color analysis task to analyze only the region that represents the vehicle, and not the entire image.
`ColorThreshold`	The analysis task discards colors that do not make up a significant proportion of the region (as specified by this parameter).
`ColorSpace`	The color space in which the results are provided (**`RGB`**, **`YCbCr`**, **`HSL`**, **`HSV`**, **`CIELAB`**).

For example:

```
[VehicleColor]
Type=ColorCluster
Input=VehicleModel.ColorRegionWithSource
RestrictToInputRegion=TRUE
ColorThreshold=20
ColorSpace=HSV
```

For more information about the parameters that customize color analysis, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 18: Scene Analysis

Scene analysis detects important events that occur in video. You can use scene analysis to monitor video streamed from CCTV cameras, to assist you with the detection of potential threats, illegal actions, or alert you to situations where help is required. The Scene Analysis engine can be trained to detect whatever you require.

Typical examples of objects and events you might want to detect in CCTV footage include:

- Abandoned bags or vehicles.
- Zone breaches and trip wire events, for example a person entering a restricted area.
- A vehicle breaking traffic laws or running a red light.
- Pedestrian or vehicular traffic congestion.

Train the Scene Analysis Engine

To run scene analysis, you must create a training configuration that describes the events that you want to detect. To create the training configuration, use the Scene Analysis Training Utility.

You can use the Training Utility to:

- Define one or more regions of interest in the scene.
- Mask parts of the scene that you do not want to monitor.
- Define the size, shape, orientation, velocity, and color of the objects that you want to detect, and the permitted variations in all of these properties.
- Define the position of traffic lights in the scene, so that Media Server can read the lights.
- Display the video being analyzed by Media Server so that you can confirm objects are being tracked correctly.
- Set up filters to reduce the number of false alarms. For example, you might want Media Server to generate alarms only for objects that are larger than a particular size.
- Review the alarms that have been generated using your training configuration, and classify those alarms as suspicious (true alarms) or not suspicious (false alarms). The Training Utility can then suggest improvements to the configuration to minimize the number of false alarms and the number of missed alarms.

While you are training Media Server, you can use the Training Utility to start and stop ingestion and analysis (`process` actions). When you start processing through the Training Utility, Media Server makes video frames and alarm data available to the Training Utility. The Training Utility needs this data during the training process.

After you have finished training, send your finished configuration to Media Server. To run Scene Analysis in a production environment, do not start processing through the Training Utility. Instead,

create a new configuration containing a scene analysis task, and start the `process` action as described in "Start Processing" on page 93.

For more information about using the Scene Analysis Training Utility, refer to the documentation for the Training Utility.

Run Scene Analysis

While you are training Media Server, you can use the Training Utility to start and stop ingestion and analysis (`process` actions). However, to run scene analysis in a production environment, do not start processing through the Training Utility. Instead, use the following procedure to create a configuration that contains a scene analysis task, and start the `process` action as described in "Start Processing" on page 93.

To detect important events in video

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=ISAS
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`SceneAnalysis`**.
`TrainingCfgName`	The name of the training configuration file to use to detect important events. This is the name of the file that you created using the Scene Analysis Training Utility.

   For example:

   ```
   [ISAS]
   Type=SceneAnalysis
   TrainingCfgName=redlights
   ```

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 19: Extract Keyframes

Media Server can identify the keyframes in a video. A *keyframe* is the first frame following a significant scene change. Keyframes are often used as preview images for video clips.

Configure Keyframe Extraction

To extract keyframes from video

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=Keyframes
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`keyframe`**.
`Input`	(Optional) The name of the image track to process.
`ForceAfter`	(Optional) The maximum time interval between keyframes.
`QuietPeriod`	(Optional) The minimum time interval between keyframes.
`Sensitivity`	(Optional) The sensitivity of the engine to scene changes.

   For example:

   ```
   [Keyframes]
   Type=keyframe
   Sensitivity=0.6
   ForceAfter=5minutes
   ```

   For more information about the parameters that you can use to configure this task, refer to the *Media Server Reference*.
4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 20: Detect and Read Barcodes

Media Server can detect and read barcodes that appear in media. For each detected barcode, Media Server returns the barcode type, its location in the image, and the information that it encodes.

Supported Barcode Types

Media Server can read the following types of barcode:

Codabar	Data Matrix	Industrial 2/5	UPC-A
Code-128	EAN-13	Matrix 2/5	UPC-E
Code-39	EAN-8	Patch Code	
Code-93	I25	PDF417	
Datalogic 2/5	IATA 2/5	QR	

Read Barcodes

To read barcodes

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=Barcodes
   ```

3. Create a new configuration section to contain the task settings and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`barcode`**.
`Input`	(Optional) The image track to process. If you do not specify an input track, Media Server processes the first track of the correct type that is produced by the ingest engine.

`Orientation`	(Optional) The orientation of barcodes in the ingested media. If barcodes might appear at an orientation other than upright, set this parameter to **`any`**. Media Server will automatically detect the orientation (from 90-degree rotations, not arbitrary angles).
`Region`	(Optional) To search for barcodes in a region of the image, instead of the entire image, specify the region to search.
`RegionUnit`	(Optional) By default, the `Region` parameter specifies the size and position of a region using percentages of the frame dimensions. Set the `RegionUnit` parameter to **`pixel`** if you want to use pixels instead.

For example:

```
[Barcodes]
Type=barcode
Orientation=any
```

For more information about the parameters that you can use to configure this task, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example Barcode Task Configuration

To detect and read barcodes, you could use the following task configuration:

```
[Barcodes]
Type=barcode
BarcodeTypes=all
Orientation=any
```

Setting the `BarcodeTypes` parameter to **`all`** detects all types of barcodes, including QR codes.

By default Media Server only detects barcodes that are upright, but setting the `Orientation` parameter to **`any`** means that Media Server can also detect barcodes that have been rotated by 90 or 180 degrees.

Barcode Analysis Results

The following XML is a single record produced during barcode analysis:

```
<record>
  ...
  <trackname>barcode.Result</trackname>
  <BarcodeResult>
    <id>b8c4331e-6058-4786-83d9-a43e605f463e</id>
    <text>some text</text>
    <barcodeType>Code-128</barcodeType>
    <region>
      <left>94</left>
```

```
      <top>66</top>
      <width>311</width>
      <height>98</height>
    </region>
  </BarcodeResult>
</record>
```

The record includes the following information:

- The `id` element provides a unique identifier for the detected barcode. The barcode analysis engine issues an ID for each detected appearance of a barcode. If you are detecting barcodes in video and consecutive frames show the same barcode in a near-identical location, all records related to that appearance will have the same ID.

  For example, if a barcode appears in the same location for fifty consecutive video frames, the engine uses the same ID for each record in the data track and produces a single record in the result track. The record in the result track will have a timestamp that covers all fifty frames.

  If the barcode moves to a different location on the screen, or disappears and then reappears, the engine considers this as a new detection and produces a new ID and a new record in the result track.
- The `text` element contains the data encoded by the barcode.
- The `barcodeType` element contains a string which describes the type of the detected barcode. This can be any of the values listed in "Supported Barcode Types" on page 190.
- The `region` element describes the position of the barcode in the ingested media.

Chapter 21: Analyze Colors

Media Server can identify the dominant colors in images and video frames, or in a region of an image or video frame. The region can be manually defined in the configuration, or supplied by another analysis task. Media Server clusters similar colors and returns the color at the center of each cluster as a value in the selected color space (for example, RGB). It also returns the proportion of the pixels in the frame that belong to each cluster.

Perform Color Analysis

To configure color analysis

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task using the `AnalysisEngine` parameter. You can give the task any name, for example:

   ```
   [Analysis]
   ...
   AnalysisEngine2=ColorClusterTask
   ```

3. Create a new configuration section to contain the task settings and set the following parameters:

`Type`	The analysis engine to use. Set this to **`ColorCluster`**.
`Input`	(Optional) The name of the track that contains the images to process. If you do not specify an input track, Media Server processes the first track of the correct type.
`ColorSpace`	(Optional) The color space in which the results of analysis are provided.

   For example:

   ```
   [ColorClusterTask]
   Type=ColorCluster
   Input=MyTask.ResultWithSource
   ColorSpace=RGB
   ```

4. (Optional) You can restrict color analysis to a specific region of the image or video frame.
   - To restrict analysis to a region that you define manually, add the following configuration parameters to the task:

`Region`	The region to analyze. Specify the region using a comma-separated list of

values that describe a rectangle (*left,top,width,height*).

`RegionUnit` The units you want to use to define the region (`pixel` or `percent`).

- To restrict analysis to a region that is supplied by another analysis task, add the following configuration parameter:

  `RestrictToInputRegion` A Boolean value that specifies whether to analyze a region of the input image or video frame that is specified in the input record, instead of the entire image. Set this parameter to **TRUE**.

  Note: If you set `RestrictToInputRegion`, the input track that you specify must contain region data. Many analysis engines, for example clothing detection or object recognition, produce records that contain regions.

5. Save and close the configuration file.

Color Analysis Results

The following XML shows a single record produced by color analysis.

```
<output>
  <record>
    ...
    <trackname>ColorCluster.Result</trackname>
    <ColorClusterResult>
      <id>76b7b8dd-59f6-4fe7-9a6e-bcfae3cf94e8</id>
      <colorspace>RGB</colorspace>
      <cluster>
        <color>232 242 252</color>
        <proportion>77.12</proportion>
      </cluster>
      <cluster>
        <color>237 28 36</color>
        <proportion>21.54</proportion>
      </cluster>
    </ColorClusterResult>
  </record>
</output>
```

The record contains the following information:

- The `colorspace` element specifies the color space in which the results are provided. You can choose the color space by setting the `ColorSpace` configuration parameter.
- Each `cluster` element represent a cluster of similar colors.
  - The `color` element provides the color at the center of the cluster, in the color space requested.
  - The `proportion` element specifies the percentage of the image or video frame that belongs to that cluster.

Chapter 22: Generate Image Hashes

Media Server can generate an image hash that describes the approximate color distribution of an image or video frame.

Introduction

An image hash describes the approximate color distribution of an image, in the form of text that you can index into HPE IDOL Server. You can use image hashes to quickly identify duplicates in a set of images, because identical images will produce the same hash. Images that have been resized will also produce the same hash, but other transformations such as translation or perspective distortion will cause the hash to change.

Set up a Task to Generate Image Hashes

To generate image hashes

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Analysis]` section, add a new analysis task by setting the `AnalysisEngine`*N* parameter. You can give the task any name, for example:

   ```
   [Analysis]
   AnalysisEngine0=ImageHash
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`ImageHash`**.
`Input`	(Optional) The name of the track to generate image hashes from.
`Region`	(Optional) To create a hash from a region of the image, instead of the entire image, specify the region.
`RegionUnit`	(Optional) The units used to specify the position and size of the region of interest (**`pixel`** or **`percent`**).

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example Configuration

To ingest an image file, produce an image hash, and output the response to the ACI response, you could use the following configuration:

```
// ======================= Ingest =======================
[Ingest]
IngestEngine = SingleImages

[SingleImages]
Type = Image

// ======================= Analysis =======================
[Analysis]
AnalysisEngine0 = ImageHash

[ImageHash]
Type = ImageHash

// ======================= Output =======================
[Output]
OutputEngine0 = ACI

[ACI]
Type=response
Input=ImageHash.result
```

Image Hash Results

Media Server returns a text hash of the image or video frame, suitable for indexing into HPE IDOL Server. For example:

```
<record>
  <pageNumber>1</pageNumber>
  <trackname>imagehash.Result</trackname>
  <ImageHashResult>
    <imageHash>
         AAAAD AAAAD BAAAD BAAAD CAAAD CAAAD NIAAD
         DAFCC OIKDC DAAAD OIAAD EAAAD PIAAD EAAAD
         PIAAD FAAAD QIAAD FAAAD QIAAD GAAAD RIAAD
         ...
    </imageHash>
  </ImageHashResult>
</record>
```

Part IV: Encode Media

This section describes how to encode media using Media Server.

- "Encode Video to a File or UDP Stream"
- "Encode Video to a Rolling Buffer"
- "Encode Images to Disk"

Chapter 23: Encode Video to a File or UDP Stream

This section describes how to encode video to a file, or to a UDP stream.

Introduction

Media Server can encode the video it ingests and write the video to files, to a stream, or to a rolling buffer.

There are several reasons for encoding video:

- If you are ingesting video from a stream, you can encode the video for playback at a later time. If your analysis tasks detect interesting events in the video, you might want to review those events.
- You can choose the size and bit rate of the encoded video. If you are analyzing sources such as high-definition television broadcasts, you might want to reduce the size and bit rate of the video for storage and streaming to users when they search for clips.

Encoding video does not affect the video frames used for analysis. Analysis always uses the source video.

Media Server provides several audio and video profiles that contain settings for encoding. For example, there are profiles for high-quality output, which you can use if you want to prioritize quality over disk space. By default, the files are stored in the `profiles` folder in the Media Server installation directory. You should not need to modify the profiles. When you configure encoding, you can select the profiles that you want to use.

Encode Video to MPEG Files or a UDP Stream

To encode video to an MPEG file or UDP stream, follow these steps.

To encode video to a file or stream

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Encoding]` section, add a new encoding task by setting the `EncodingEngine`*N* parameter. You can give the task any name, for example:

   ```
   [Encoding]
   EncodingEngine0=MyEncodingTask
   ```

3. Below the `[Encoding]` section, create a configuration section for the engine by typing the task

name inside square brackets. For example:

```
[MyEncodingTask]
```

4. In the new section, set the following parameters:

`Type`	The encoding engine type. Set this parameter to **`mpeg`**.
`VideoSize`	(Optional) The size of the encoded video, in pixels, width followed by height. If you want to use the same dimensions as the source, set this parameter to **`copy`**.

To write the encoded video to an MPEG file, set the following parameters:

`OutputPath`	The path of the encoded output file(s). You can use macros to create output paths based on the information contained in the encoded records.
`UrlBase`	The base URL that will be used to access the encoded files. This is used when Media Server generates proxies. You can use macros to create the URL base from information contained in the encoded records.
`Segment`	(Optional) Specifies whether to split the output file into segments.
`SegmentDuration`	(Optional) The maximum duration of a segment, in seconds.

To generate a live UDP stream of the content that Media Server is ingesting, set the following parameters:

`OutputURL`	The UDP output stream. For example, `udp://239.255.1.123:4321`.
`Format`	The container format. For example, `mpegts` for MPEG transport stream.

For example:

```
[MyEncodingTask]
Type=mpeg
OutputPath=./mp4/%year%/%month%/%day%/%timestamp%_%segment.sequence%.mp4
UrlBase=http://www.mysite.com/video/clips/mp4/%year%/%month%/%day%/
```

For more information about the configuration parameters and macros that you can use, refer to the *Media Server Reference*.

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 24: Encode Video to a Rolling Buffer

This section describes how to configure encoding to a rolling buffer.

Store Video in a Rolling Buffer

Media Server can save a copy of the video it ingests to a rolling buffer. A *rolling buffer* is a fixed amount of storage where the oldest content is discarded to make space for the latest.

Media Server can write video directly into the rolling buffer, producing an evidential copy that has not been modified in any way. You might require an evidential copy for legal reasons. Media Server can also encode video before writing it to the buffer, creating a non-evidential copy that is optimized for storage and playback.

Note: Evidential mode is not supported when the source video is in MJPEG format.

A rolling buffer is useful for both broadcast monitoring and surveillance applications. For example, in a surveillance application you could configure the rolling buffer with sufficient storage to contain all the video captured by a camera in the last 7 days. If an event occurs you can play the content from the rolling buffer and view the video around the time of the event. If you needed to document the event, you would have 7 days to extract images and video from the rolling buffer before the video is overwritten.

Rolling buffers are configured in a separate configuration file (not in the Media Server configuration file). The rolling buffer configuration file contains settings such as the paths to the storage locations on disk, and the amount of storage to allocate to each rolling buffer.

You can configure as many rolling buffers as you need. For example, you could choose to save an evidential copy of the ingested video to one rolling buffer and a smaller compressed copy to another. You might also want to set up multiple rolling buffers if you ingest video from separate cameras or channels. For example, if you ingest 12 hours of video from one channel and then 48 hours from another you can use multiple rolling buffers to store the last 12 hours from each channel.

Video is served from the rolling buffer using the HTTP Live Streaming (HLS) protocol. An HLS-compliant media player can request an HLS playlist from Media Server or the HPE MMAP REST endpoint. The playlist points to video segments in the rolling buffer, and these segments are served by

an external component such as a Web server or HPE MMAP. Media Server does not include a Web server.

Calculate Storage Requirements

When setting up a rolling buffer, consider what you intend to store and determine the amount of storage you need. It is important to allocate sufficient storage space because as soon as the buffer is full, Media Server overwrites the oldest video to make space for the latest. To avoid losing important data, ensure you have sufficient capacity.

The amount of storage you should allocate to a rolling buffer depends on:

- the amount of time for which you want to keep video before it is overwritten.
- the bitrate of the video you encode to the rolling buffer.

For example, if you intend to store video for 1 week, and you encode the video at 4 megabits per second, you would need approximately 350GB of disk space.

After setting up your rolling buffer, HPE recommends that you check that the buffer is storing as much video as you expected.

Set Up Rolling Buffers

The settings for your rolling buffers are stored in a separate configuration file. This is so that you can share the rolling buffer configuration between multiple Media Servers . For example, you might have one Media Server writing video to a rolling buffer and another generating playlists.

A default rolling buffer configuration file is included with Media Server in the `/encoding/rollingBuffer` folder.

In `rollingBuffer.cfg`, you can control the following settings:

- the location of the root folder for each rolling buffer.
- the maximum number and size of the files allocated to each rolling buffer. The number of files multiplied by their size gives the total amount of storage for a rolling buffer. Media Server saves video across multiple files because this is beneficial to disk performance. You can set a default value for all rolling buffers and override it for individual rolling buffers.
- the prefixes added to URLs used in playlists.

Note: Apart from the settings in the `[Defaults]` section, do not edit the rolling buffer configuration file in a text editor. To add or modify rolling buffers, use the ACI actions provided by Media Server.

To set the path to the rolling buffer configuration file

1. Open the Media Server configuration file and find the `[Paths]` section.
2. Ensure that the path to the rolling buffer configuration file is set, for example:

```
[Paths]
RollingBufferConfigPath=./encoding/rollingBuffer/rollingBuffer.cfg
```

Relative paths must be relative to the Media Server working directory. If you share a rolling buffer configuration file between multiple Media Servers, specify a UNC path.

3. Save and close the configuration file.
4. If you made any changes to the configuration file, restart Media Server.

To configure default settings for rolling buffers

1. Open the rolling buffer configuration file.
2. In the `[Defaults]` section, set the following parameters:

`RootPath`	The path of the directory to store the rolling buffer files in. Relative paths must be relative to the rolling buffer configuration file.
`MaxFiles`	The maximum number of files for each rolling buffer.
`MaxFileSizeMB`	The maximum size in MB for each file.
`MediaSegmentTemplate`	(Optional) A template to use to construct the URL of every media segment in a playlist.
`VariantSegmentTemplate`	(Optional) A template to use to construct the URL of every `GetPlaylist` action that is produced by Media Server.

For example:

```
[Defaults]
RootPath=C:\VideoRecording\
MaxFiles=10
MaxFileSizeMB=100
```

For more information about these configuration parameters, refer to the *Media Server Reference*.

To add a new rolling buffer

- Use the ACI action `AddStream`, for example:

  ```
  /action=AddStream&name=NewsChannel&maxfiles=10&maxfilesize=100
  ```

  where the `name` parameter is required and specifies the name of the rolling buffer. The other parameters are optional and override the default rolling buffer settings.

To remove a rolling buffer

Caution: This deletes all video that has been stored in the rolling buffer.

- Use the ACI action `RemoveStream`, for example:

  ```
  /action=RemoveStream&name=NewsChannel
  ```

  where the `name` parameter is required and specifies the name of the rolling buffer to remove.

For more information about the actions that you can use to configure rolling buffers, refer to the *Media Server Reference*.

Pre-Allocate Storage for a Rolling Buffer

Media Server must allocate storage for a rolling buffer before it can write encoded video to the buffer. Allocating storage is not instantaneous so to ensure that Media Server can start recording from a stream immediately, pre-allocate storage before you start a session.

To pre-allocate storage in the rolling buffer

- Send the `PreAllocateStorage` action to Media Server:

  `http://localhost:14000/action=PreAllocateStorage`

  For more information about this action, refer to the *Media Server Reference*.

Write Video to a Rolling Buffer

To write ingested video to a rolling buffer, follow these steps.

To write video to a rolling buffer

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Encoding]` section, add a new encoding task and a configuration section for the task:

   ```
   [Encoding]
   EncodingEngine0=RollingBufferEvidential

   [RollingBufferEvidential]
   ```

3. In the new section, set the following parameters:

`Type`	The type of engine to use. Set this parameter to **`RollingBuffer`**.
`Stream`	The name of the rolling buffer to write video to. You must have created the rolling buffer (see "Set Up Rolling Buffers" on page 202).

4. Decide whether to store an evidential copy of the video or encode the video so that it is optimized for storage and playback:
   - To store an evidential copy of the video, set the following parameter:

`EvidentialMode`	To write an evidential copy of the video to the rolling buffer, set **`EvidentialMode=true`**. Media Server ignores any encoding settings defined for this task, but you must also set **`AudioInput=none`** and **`ImageInput=none`**.

   - To store an encoded (non-evidential) copy of the video, set the following parameters:

`AudioProfile`	The audio encoding profile to use.
`VideoProfile`	The video encoding profile to use.

For example:

```
[Encoding]
EncodingEngine0=RollingBufferEvidential
EncodingEngine1=RollingBufferCompressed

[RollingBufferEvidential]
Type=RollingBuffer
Stream=evidential
EvidentialMode=true
ImageInput=none
AudioInput=none

[RollingBufferCompressed]
Type=RollingBuffer
Stream=compressed
AudioProfile=mpeg4audio
VideoProfile=mpeg4video_h264_sd
```

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

View the Rolling Buffer Contents

To retrieve a list of all the rolling buffers that have been configured use the following procedure.

To list your rolling buffers

- Send the `GetStreams` action to Media Server. For example:

  ```
  http://localhost:14000/action=GetStreams
  ```

  Media Server returns a list of all of your rolling buffers.

You can also retrieve information about the video stored in a rolling buffer. For example, you might record video for one hour each morning and two hours each afternoon. To get information about what has been stored in a rolling buffer, use the following procedure.

To return a list of the content in a rolling buffer

- Send the `GetRecordedRanges` action to Media Server.

  You can filter the results by setting the following optional parameters:

`Stream`	The rolling buffer to list of recorded content for. If this parameter is not set, Media Server returns results for all rolling buffers.
`StartTime`	Only show content available after this time. Specify the start time in epoch

milliseconds or ISO-8601 format.

`Duration` The length of time after the start time to return a list of available content for. Specify the duration in milliseconds.

For example, to retrieve a list of content available in the `BBCNews` rolling buffer for 24 hours from 16:27:54 on 21 February 2014:

```
http://localhost:14000/action=GetRecordedRanges&Stream=BBCNews
                                                &StartTime=1393000074243
                                                &Duration=86400000
```

Retrieve an HLS Playlist

Media Server generates HTTP Live Streaming (HLS) playlists that can be used by a media player to request content from a rolling buffer. To play video from a rolling buffer, your system must meet the following requirements:

- You must configure a Web server or use the HPE MMAP REST endpoint to serve video segments from your file system to the media player.
- The media player that you use must be HLS-compliant. By default, Media Server generates HLS version 4 playlists, but you can also configure Media Server to generate HLS version 1 playlists.

To retrieve a playlist

- Send the `GetPlaylist` action to Media Server. Set the following parameters:

  `Stream` The name of the rolling buffer to request video from.

  `StartTime` (Set this parameter or `Offset`) The start time of the playlist in ISO 8601 format or epoch milliseconds.

  `Offset` (Set this parameter or `StartTime`) Specifies the start time of the playlist by calculating an offset from the current time. For example, if you specify an offset of one hour, the start time for the playlist is one hour ago. Specify the offset in milliseconds.

  `Duration` (Optional) The length of time after the start time that the playlist covers. Specify the duration in milliseconds.

  `HLSVersion` (Optional) By default, Media Server generates HLS version 4 playlists. To obtain an HLS version 1 playlist set this parameter to `1`.

  For example, to retrieve a playlist that contains five minutes of content from the `BBCNews` rolling buffer, starting from 16:27:54 on 21 February 2014:

```
http://localhost:14000/action=GetPlaylist&Stream=BBCNews
                                         &StartTime=2014-02-21T16:27:54Z
                                         &Duration=300000
```

To retrieve a playlist that contains content from the `BBCNews` rolling buffer, starting from 16:27:54 on 21 February 2014, with no end time, use the following action. If you play the content at normal speed and there is no break in recording, the media player could continue playing content forever:

```
http://localhost:14000/action=GetPlaylist&Stream=BBCNews
                                          &StartTime=2014-02-21T16:27:54Z
```

Media Server returns the playlist. If you open the playlist with an HLS-compliant media player, the player will play the video from the rolling buffer.

Create a Clip from a Rolling Buffer

Use the following procedure to retrieve a section of video from the rolling buffer.

To create a clip from a rolling buffer

- Send the `CreateClip` action to Media Server. Set the following parameters:

`Stream`	The name of the rolling buffer to create the clip from.
`StartTime`	The start time of the clip in epoch-milliseconds or ISO-8601 format. Video must exist in the rolling buffer for the start time that you specify (or begin within 15 seconds of the start time). If video begins within 15 seconds after the start time that you specify, Media Server automatically adjusts the start time. If there is no video in the rolling buffer within 15 seconds after the start time, the action fails.
`Duration`	The duration of the clip in milliseconds.
`OutputFormat`	(Optional) The format of the container file that is returned (default **`ts`**, but you can also choose **`mp4`**).
`Path`	(Optional) The path to save the clip to. The directory must be on a file system that Media Server can access. If you do not set this parameter, the file is returned in the response.

For example,

```
http://localhost:14000/action=CreateClip&Stream=BBCNews
                                        &StartTime=1393000074243
                                        &Duration=300000
                                        &Path=./temp/News1.ts
```

This action instructs Media Server to create a five minute clip from the rolling buffer `BBCNews`, beginning from Fri, 21 Feb 2014 16:27:54 GMT, and to save the clip as the `News1.ts` file in the `temp` directory.

Create an Image from a Rolling Buffer

To obtain a single video frame from a rolling buffer, use the following procedure.

To create an image from a rolling buffer

- Send the `CreateImage` action to Media Server. Set the following parameters:

  `Stream` The name of the rolling buffer to create the image from.

  `Time` The time that the desired frame occurs in the rolling buffer. Specify the time in epoch-milliseconds or ISO-8601 format.

  For example,

```
http://localhost:14000/action=CreateImage&Stream=BBCNews
                                          &Time=1393000074243
```

Use Multiple Media Servers

It is possible to configure multiple Media Servers to use the same rolling buffer configuration file. You can have a maximum of one Media Server writing video into a rolling buffer, but other Media Servers can read content from the rolling buffer.

If you configure multiple Media Servers to use the same rolling buffer configuration file, you must:

- grant read access to the rolling buffer configuration file to all of the Media Servers.
- grant write access to the rolling buffer configuration file to only one Media Server. If you need to change the settings for your rolling buffers, you must send actions to this Media Server.
- allow only one Media Server to write video into each rolling buffer. The `Stream` parameter in a rolling buffer encoding task specifies which rolling buffer to write video to.

Chapter 25: Encode Images to Disk

This section describes how to encode images to disk.

Introduction

Media Server can encode images to disk.

There are several reasons you might want to do this:

- To use the images in a front-end application. For example, if you deploy Media Server for broadcast monitoring purposes, you might extract keyframes from the ingested video and use these as thumbnails in a web application that presents the analysis results.
- To obtain training images. For example, if you run face detection on a video, you can write an image of each detected face to disk. You could then use these images for training face recognition.

You can encode images in one of several formats. If you intend to use the images for training, HPE recommends using a format that is not compressed, or uses lossless compression. PNG is a good choice for training images. The JPEG image format uses lossy compression to reduce file size and, as a result, the quality of the image is reduced. This makes JPEG images unsuitable for training but the smaller file size is advantageous if you intend to use the images in a front-end that might be viewed over the web.

Encode Images

To encode images

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Encoding]` section, add a new encoding task by setting the `EncodingEngine`*N* parameter. You can give the task any name, for example:

   ```
   [Encoding]
   EncodingEngine0=KeyframeImages
   ```

3. Below the `[Encoding]` section, create a configuration section for the engine by typing the task name inside square brackets. For example:

   ```
   [KeyframeImages]
   ```

4. In the new section, set the following parameters:

`Type`	The encoding engine to use. Set this parameter to **`ImageEncoder`**.

`ImageInput`	The name of the track that contains the images to encode.
`OutputPath`	The path and file name for the output files. If the directory does not exist, Media Server creates it. You can use macros to create output paths based on the information contained in a record.
`UrlBase`	The URL that will be used to access the encoded files.
`ImageSize`	(Optional) The output image size in pixels, width followed by height. If you do not set this parameter, Media Server uses the original image size. If you specify only one dimension, Media Server calculates the other, maintaining the original aspect ratio. For example, to specify a width of 300 pixels and have Media Server calculate the appropriate height, set this parameter to **`ImageSize=300x0`**. Setting this parameter only modifies the size of the encoded image. The image encoder does not scale any metadata that describes the position of an object in the image. To scale images and position metadata, consider using a scale transformation task.
`FrameRateMax`	(Optional) The maximum number of images to encode per second of video. This configuration parameter is ignored if the source media is an image file or document.

For example:

```
[KeyframeImages]
Type=ImageEncoder
ImageInput=myKeyframeEngine.ResultWithSource

OutputPath=c:\output\keyframes\%record.starttime.year%-%record.starttime.month%
-%record.starttime.day%\%record.starttime.timestamp%.jpg

UrlBase=http://www.mysite.com/keyframes/%record.starttime.year%-%record.startti
me.month%-%record.starttime.day%/
ImageSize=192x108
```

For more information about the parameters that you can use to configure the image encoder, refer to the *Media Server Reference*.

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Part V: Event Stream Processing

This section describes how to use Event Stream Processing (ESP). You can use ESP to introduce custom logic into your processing tasks.

Chapter 26: Event Stream Processing

This section describes event stream processing in Media Server.

Introduction to Event Stream Processing

Event Stream Processing (ESP) applies rules to the output of other tasks, including other ESP tasks.

One use of ESP is to identify interesting events from a large number of records. A simple example is detecting the occurrence of a particular word in an audio stream. A more complex example is detecting when a combination or sequence of events occurs within a specific time interval; for example, a number plate is detected shortly after a traffic light changes to red.

The tracks passed to ESP engines are not modified. An ESP task produces a new output track that contains records which meet the specified conditions.

ESP engine	Description	Example
Filter	Filters a track, and produces an output track that contains only records that meet specified conditions.	Filter speech-to-text results for a news channel to produce an output track that only contains records for the word "weather".
Deduplicate	Identifies duplicate records in a track, and produces an output track without the duplicates.	Face recognition produces a result each time a person is recognized. Deduplication can filter the output to remove any records produced when the same person is recognized again within a certain number of seconds.
Or	Combines two or more tracks into a single output track. The output track contains all of	Combine tracks from OCR,

	the records from all of the input tracks.	speech-to-text, and face recognition to produce an output track that contains information about text, speech, and faces in a video.
And	Compares two tracks to identify combinations of events. The event in the second track must occur within a specific time interval (before or after) the event in the first track. The records in the output track each contain a pair of related records. A record in the first track can appear in the output track more than once if it becomes part of multiple combinations.	Identify when the text "election results" appears on-screen and a news presenter speaks the name of a politician up to ten seconds before or after the text appears.
AndThen	Compares two tracks to identify combinations of events. The event in the second track must occur at the same time as, or within a specific time interval after the event in the first track. The records in the output track each contain a pair of related records. A record in the first track can appear in the output track more than once if it becomes part of multiple combinations.	Identify when the text "election results" appears on-screen and a news presenter speaks the name of a politician up to ten seconds after the text appears.
AndNot	Compares two tracks to identify when an event occurs in the first track and nothing happens in the second track, within a specific time interval (before or after) that event. The records in the output track contain the record from the first track. Records from the second track are never included in the output.	Produce a track containing records for all appearances of a logo that are not accompanied by the company name in the ten seconds preceding or following the logo's appearance.
AndNotThen	Compares two tracks to identify when an event occurs in the first track and nothing happens in the second track, within a specific time interval after that event. The records in the output track contain the record from the first track. Records from the second track are never included in the output.	Produce a track containing records for all appearances of a logo that are not followed within ten seconds by the company name.
Combine	Compares two tracks to identify combinations of events. This is similar to the And ESP task, except: • It produces an output record for every record in the first input track, and this output record contains copies of all related records from the second input track. In comparison, the And task creates one output record for each pair of related	Combine information about detected faces that appear at the same time so that all of the faces can be blurred by a transformation task.

	records. • Every record from the first input track always appears in the output, even if there are no related records in the second input track.	

The ESP engines support configuration parameters that allow you to customize the operations for your data. Some engines also allow you to run scripts written in the Lua scripting language. For more information about Lua scripts, see "Write a Lua Script for an ESP Engine" on page 228.

ESP engines can accept any track. They also accept the output of other ESP engines.

Event Stream Processing with Documents

Event stream processing can be used to discover combinations or sequences of events that occur in video. If you are processing images or documents you can use event stream processing to determine when interesting records occur on the same page.

Note: The `AndThen` and `AndNotThen` tasks are not supported for image and document processing.

When you process images and documents, the ESP time interval parameters are ignored. Instead, Media Server considers that records are related if they are related to the same page of an image or document. PDF files are a special case and records are considered to be related if they relate to the same image element.

Filter a Track

You can filter a track to extract records that match particular conditions. For example, you can:

- extract records from OCR analysis results that match or contain a specified string
- extract records from a speech-to-text task that match or contain a specified string
- extract records from a face detection task that describe faces that appear in a particular region of the frame
- extract records from an object recognition task that match a specific object

To filter a track

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[EventProcessing]` section, add a new task by setting the `EventProcessingEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [EventProcessing]
   EventProcessingEngine0=Weather
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type` The ESP engine to use. Set this parameter to **`filter`**.

`Input` The output track, produced by another Media Server task, that you want to filter.

4. Set one of the following parameters to specify how the input track is filtered:

   `RequiredString` A string that a record must match to be included in the output track. (The input track must contain text data).

   `RequiredSubString` A string that a record must contain to be included in the output track. (The input track must contain text data).

   `LuaScript` The name of a Lua script that defines conditions that a record must meet in order to be included in the output track from the ESP engine. For more information, see "Write a Lua Script for an ESP Engine" on page 228.

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following configuration produces a new track called `Weather.Output`. This track only contains records that include the word "weather".

```
[EventProcessing]
EventProcessingEngine0=Weather

[Weather]
Type=filter
Input=speechtotext.result
RequiredSubString=weather
```

Deduplicate Records in a Track

Deduplication identifies duplicate records in a track, and produces a new track that has the duplicate records removed.

The engine identifies two identical records that occur within a specific time interval and discards the second record. For example, face recognition can produce a record for each frame that a person is recognized in. The Deduplicate ESP engine can remove duplicate records so that the track contains a single record for each recognized person.

You can specify the conditions that make two records identical. There are several options:

- any records are considered identical; Media Server discards any record that occurs within the minimum time interval of the first record
- use the default equivalence conditions for the track. Each type of track has its own default equivalence conditions; for example, OCR records are considered equivalent if the text is identical. The table lists the equivalence conditions for each output track.

Analysis engine	Output tracks	Equivalence conditions

Barcode	Data, DataWithSource, Result, ResultWithSource	Text field must be identical.
Face detection	Data, DataWithSource, Result, ResultWithSource	Rectangle field must be identical.
Face demographics	Result, ResultWithSource	All custom fields must be identical.
Face recognition	Result, ResultWithSource	Database and identifier fields must be identical.
Face state	Result, ResultWithSource	All custom fields must be identical.
Numberplate	Data, DataWithSource, Result, ResultWithSource	Text field must be identical.
	PlateRegion	Polygon field must be identical.
Object detection	Result, ResultWithSource	The detector, classification identifier, and region must be identical.
Object recognition	Data, DataWithSource, Result, ResultWithSource	Database and identifier fields must be identical.
Image classification	Result, ResultWithSource	Classifier and identifier fields must be identical.
OCR	Data, DataWithSource, Result, ResultWithSource	Text field must be identical.
SceneAnalysis	Data, DataWithSource, Result, ResultWithSource	All custom fields must be identical.
SpeakerID	Result	Text field must be identical.
SpeechToText	Result	Text field must be identical.

- specify equivalence conditions using a Lua script. For example, you might want to declare two Face records identical if they contain the same person, even if the location of the face in the frame is different. For more information about Lua scripts, see "Write a Lua Script for an ESP Engine" on page 228.

To deduplicate a track

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[EventProcessing]` section, add a new task by setting the `EventProcessingEngine`*`N`* parameter. You can give the task any name, for example:

```
[EventProcessing]
EventProcessingEngine0=Deduplicate
```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The ESP engine to use. Set this parameter to **`deduplicate`**.
`Input`	The name of the track to deduplicate. This must be an output track produced by another task.
`MinTimeInterval`	The minimum time in milliseconds between records. When you process video, the engine only discards duplicate records that occur within this time interval. If you are processing images or documents this parameter is ignored.
`PredicateType`	(Optional) The conditions to use to determine whether two records are considered identical. You can set one of: • `always`. Any records are considered identical. • `default`. Use the default equivalence conditions for the track type. • `lua`. Use the conditions defined in a Lua script specified in the `LuaScript` parameter.
`LuaScript`	(Optional) The name of a Lua script that determines whether two records are considered identical. For more information, see "Write a Lua Script for an ESP Engine" on page 228. If this parameter is set, Media Server uses the Lua script to determine whether records are identical, regardless of the `PredicateType` parameter setting.

   For more information about these parameters, including the values that they accept, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following example deduplicates the output track from an OCR task by discarding all identical records that occur within 1 second after a record. The records are judged to be identical based on the default equivalence conditions for the OCR track (the text is identical).

```
[EventProcessing]
EventProcessingEngine0=DeduplicateOCR

[DeduplicateOCR]
Type=deduplicate
Input=myocr.data
MinTimeInterval=1000
PredicateType=default
```

Combine Tracks

You can combine two or more tracks into a single track. The "Or" ESP engine creates an output track that contains the records from all of the input tracks. Each record in the resulting track includes the name of the track that it originated from.

For example, you could combine output tracks from speech-to-text and face recognition. The records in the resulting track would contain a transcript of speech in the video, or details of recognized faces.

Tip: This engine combines tracks, not records.

To combine tracks

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[EventProcessing]` section, add a new ESP task by setting the `EventProcessingEngine`*N* parameter. You can give the task any name, for example:

   ```
   [EventProcessing]
   EventProcessingEngine0=Combine
   ```

3. Create a new configuration section to contain the task settings, and set the following parameters:

`Type`	The ESP engine to use. Set this parameter to **`or`**.
`Input`*`N`*	The names of the tracks that you want to output as a single track. Specify two or more tracks that are produced by other tasks.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following configuration combines output tracks from a Barcode task and an OCR task:

```
[EventProcessing]
EventProcessingEngine0=Combine

[Combine]
Type=or
Input0=mybarcode.result
Input1=myocr.result
```

This task produces a new track, named `Combine.Output`, that contains all of the records from `mybarcode.result` and `myocr.result`.

Identify Time-Related Events in Two Tracks–And Engine

The *And* ESP engine compares two tracks to identify combinations of events. The engine produces an output track containing the identified record pairs.

Note: To detect events in the second track that occur only after (or at the same time as) events in the first track, use the *AndThen* engine. For more information, see "Identify Time-Related Events in Two Tracks–AndThen Engine" on page 222.

For example, you might want to identify all the times that a specific person appears in conjunction with a specific product. There are several ways you can accomplish this, but all involve the And ESP engine:

- Set up the analysis engines so they produce output tracks containing only relevant events. In this case, you would configure face recognition to recognize the specified person only, and object recognition to recognize the specified product only. You could then send the output tracks from the analysis tasks to the And engine to produce a track containing pairs of records that occurred at similar times.
- Filter the output tracks from the analysis engines, before sending the filtered tracks to the And engine. In this case, you do not need to configure face recognition and object recognition to recognize only specific faces and objects; these are extracted by Filter engines before being sent to the And engine. As with the previous method, the And engine produces a track containing pairs of records that occurred at similar times.
- Send the unfiltered output tracks from the analysis engines to an And engine that uses a Lua script to determine which events it should consider. Each time the engine detects records that occur within the specified time interval of each other, the engine runs the function in the Lua script to determine if the record pair should be included in the And output track. For more information on writing a Lua script, see "Write a Lua Script for an ESP Engine" on page 228.

To identify time-linked events in two tracks

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[EventProcessing]` section, add a new task by setting the `EventProcessingEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [EventProcessing]
   EventProcessingEngine0=AndEvents
   ```

3. Create a new configuration section for the task, and set the following parameters:

`Type`	The ESP engine to use. Set this parameter to **`and`**.
`Input0`	The first input track. This track must be an output track produced by another task.
`Input1`	The second input track. This track must be an output track produced by another task.
`MaxTimeInterval`	(Optional) The maximum difference in time (in milliseconds) between a record in the first track to a record in the second, for the records to be considered a pair. If you are processing images or documents this parameter is ignored.
`MinTimeInterval`	(Optional) The minimum difference in time (in milliseconds) between a record in the first track to a record in the second, for the records to be considered a pair. The default value is the negative of the `MaxTimeInterval` value, meaning that the event in the second track can occur before the event in the first track (up to the specified number of

milliseconds). If you are processing images or documents this parameter is ignored.

For more details about these parameters, including the values that they accept, refer to the *Media Server Reference*.

4. (Optional) To add custom logic that discards pairs of records unless they meet additional conditions, set the `LuaScript` parameter so that Media Server runs a Lua script to filter the results. For information about writing the script, see "Write a Lua Script for an ESP Engine" on page 228.

   `LuaScript` The path and file name of a Lua script to run.

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following example produces an output track named `BreakingNewsSyria.Output`. This track contains speech-to-text records that contain the word "Syria", and OCR records that match the string "Breaking News". However, the records are only included when they occur within two seconds (2000 milliseconds) of the other record type.

ESP filter tasks are used to filter the OCR and speech-to-text results, before those results are passed to the "and" ESP task.

```
[EventProcessing]
EventProcessingEngine0=BreakingNews
EventProcessingEngine1=Syria
EventProcessingEngine2=BreakingNewsSyria

[BreakingNews]
Type=filter
Input=ocr.result
RequiredString=Breaking News

[Syria]
Type=filter
Input=speechtotext.result
RequiredSubString=Syria

[BreakingNewsSyria]
Type=and
Input0=BreakingNews.output
Input1=Syria.output
MaxTimeInterval=2000
```

Identify Time-Related Events in Two Tracks–AndThen Engine

The *AndThen* ESP engine compares two tracks to identify events in the second track that occur within a specific time interval *after* (or at the same time as) events in the first track. The engine produces a track containing the identified record pairs.

Note: The AndThen engine enforces the order of events in the two tracks: events in the second track must occur after events in the first track. To detect events in two tracks that occur within a specified time interval, when the event order does not matter, use an *And* task. For more information, see "Identify Time-Related Events in Two Tracks–And Engine" on page 219.

To identify time-linked events in two tracks

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[EventProcessing]` section, add a new task by setting the `EventProcessingEngine`*N* parameter. You can give the task any name, for example:

   ```
   [EventProcessing]
   EventProcessingEngine0=AndThen
   ```

3. Create a new configuration section for the task and set the following parameters:

`Type`	The ESP engine to use. Set this parameter to **`andthen`**.
`Input0`	The first input track. This track must be an output track produced by another task.
`Input1`	The second input track. This track must be an output track produced by another task. **Note:** The events to detect in the `Input1` track must occur after, or at the same time as, the events to detect in the `Input0` track.
`MaxTimeInterval`	(Optional) The maximum time in milliseconds from the record in the first track to the record in the second track. (The engine compares the timestamp values.)
`MinTimeInterval`	(Optional) The minimum time in milliseconds from the record in the first track to the record in the second track. (The engine compares the timestamp values.) The default value is 0 (zero), meaning that the event in the second track must occur after (or at the same time as) the event in the first track.
`LuaScript`	(Optional) The name of a Lua script that defines conditions that a record pair must meet in order to be included in the output track. For information about

writing the script, see "Write a Lua Script for an ESP Engine" on page 228.

For more information about these parameters and the values they accept, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following example produces an output track (`RedLightBreach.Output`) containing records produced when a number plate is detected up to five seconds (5000 milliseconds) after traffic lights turn red. The engine takes tracks produced by SceneAnalysis and NumberPlate engines. The SceneAnalysis output track contains a record for every time the traffic lights turned red. The Numberplate output track contains a record for every number plate detected in the video.

```
[EventProcessing]
EventProcessingEngine0=RedLightBreach

[RedLightBreach]
Type=andthen
Input0=sceneanalysis.ResultWithSource
Input1=numberplate.ResultWithSource
MaxTimeInterval=5000
```

Identify Isolated Events–AndNot Engine

The *AndNot* engine compares two tracks to identify when an event occurs in the first track and nothing happens in the second track.

For example, you can identify all the occasions when a company logo appears in a video without mention of the company name in the speech:

1. An object recognition task recognizes the logo.
2. A speech-to-text task produces a transcript of the speech.
3. An ESP filter task extracts records from the speech-to-text output that contain the company name and outputs them to a new track.
4. An ESP task (using the AndNot engine) uses the output tracks from the object recognition and filter tasks. It compares events in both tracks and identifies isolated events in the first track (the object track). The engine produces an output track containing records from the object track for when a logo is recognized but is not followed within the specified time interval by the company name in the filtered SpeechToText track.

Note: The AndNot engine does not enforce an order of events in the two tracks: the first track's event is considered isolated only if an event in the second track does not occur either before or after it. If you want to consider only events in the second track occurring after (or at the same time as) events in the first track, use the *AndNotThen* engine (see "Identify Isolated Events–AndNotThen Engine" on page 225).

To identify isolated events

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[EventProcessing]` section, add a new task by setting the `EventProcessingEngine`*N* parameter. You can give the task any name, for example:

   ```
   [EventProcessing]
   EventProcessingEngine0=IsolatedEvents
   ```

3. Create a new configuration section for the task and set the following parameters:

`Type`	The ESP engine to use. Set this parameter to **`andNot`**.
`Input0`	The first input track. This track must be an output track produced by another task.
`Input1`	The second input track. This track must be an output track produced by another task.
`MaxTimeInterval`	The maximum time in milliseconds from the record in the first track to the record in the second track. (The engine compares the timestamp values.) If an event in the second track occurs within this time interval from the event in the first track, the engine discards the event in the first track. If you are processing images or documents this parameter is ignored.
`MinTimeInterval`	(Optional) The minimum time in milliseconds from the record in the first track to the record in the second track, for the two records to be considered a pair. (The engine compares the timestamp values.) The default value is the negative of the `MaxTimeInterval` value, meaning that the event in the second track can occur before the event in the first track (up to the specified number of milliseconds). If you are processing images or documents this parameter is ignored.
`LuaScript`	(Optional) The name of a Lua script that defines conditions for a discarding a record from the first track. For information about writing the script, see "Write a Lua Script for an ESP Engine" on page 228.

   For more details about the parameters, including the values that they accept, refer to the *Media Server Reference*.
4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following example produces an output track called `LogoWithoutCompanyName.Output`. This track contains records from an object recognition task that indicate when the company logo was recognized. However, the track only contains the appearances when the company name was not mentioned in the audio within five seconds.

```
[EventProcessing]
EventProcessingEngine0=FilterAudio
```

```
EventProcessingEngine1=LogoWithoutCompanyName

[FilterAudio]
Type=filter
...

[LogoWithoutCompanyName]
Type=andnot
Input0=RecognizeCompanyLogo.Result
Input1=FilterAudio.output
MaxTimeInterval=5000
```

Identify Isolated Events–AndNotThen Engine

The *AndNotThen* engine compares two tracks and produces a track containing records from the first track for events that are not followed within a specified time interval by events in the second track.

Note: The AndNotThen engine enforces the order of events in the two tracks: the first track's event is considered isolated only if an event in the second track does not occur after (or at the same time as) it. If you want to consider events in the second track occurring both before or after events in the first track, use the *AndNot* engine (see "Identify Isolated Events–AndNot Engine" on page 223).

To identify isolated events

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[EventProcessing]` section, add a new task by setting the `EventProcessingEngine`*N* parameter. You can give the task any name, for example:

   ```
   [EventProcessing]
   EventProcessingEngine0=MyAndNotThen
   ```

3. Create a configuration section for the task and set the following parameters:

`Type`	The ESP engine to use. Set this parameter to **`andNotThen`**.
`Input0`	The first input track. This track must be an output track produced by another task.
`Input1`	The second input track. This track must be an output track produced by another task.
`MaxTimeInterval`	The maximum time in milliseconds from the record in the first track to the record in the second track. (The engine compares the timestamp values.) If an event in the second track occurs within this time interval from the event in the first track, the engine discards the event in the first track.
`MinTimeInterval`	(Optional) The minimum time in milliseconds from the record in the first

	track to the record in the second track. (The engine compares the timestamp values.)
`LuaScript`	(Optional) The name of a Lua script that defines conditions for a discarding a record from the first track. For information about writing the script, see "Write a Lua Script for an ESP Engine" on page 228.

For more information about these parameters, including the values that they accept, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following example produces an output track containing records produced when a number plate is not detected within thirty seconds (30,000 milliseconds) of a barrier being raised. The AndNot engine takes tracks produced by SceneAnalysis and NumberPlate engines. The SceneAnalysis output track contains a record for every time a barrier is raised. The Numberplate output track contains a record for every number plate detected in the video. The AndNot engine output track contains all records from the SceneAnalysis output track that are not followed within thirty seconds by an event in the NumberPlate track.

```
[EventProcessing]
EventProcessingEngine0=Barrier

[Barrier]
Type=andnotthen
Input0=sceneanalysis.Result
Input1=numberplate.Result
MaxTimeInterval=30000
```

Identify and Combine Time-Related Events

The *Combine* ESP engine compares two tracks to identify combinations of events. The engine produces an output track that contains exactly one record for every record in the first input track. Each of the output records contains all related records from the second input track. The records from the first input track are output even if there are no related events in the second track.

One use case for this engine is combining records from analysis engines before running a transformation task. For example, face detection produces a record for each detected face. To blur several faces that appear simultaneously, you could combine the relevant records from face detection with the each ingested image before running the blur transformation task.

To identify and combine time-related events in two tracks

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[EventProcessing]` section, add a new task by setting the `EventProcessingEngine`*`N`* parameter. You can give the task any name, for example:

```
[EventProcessing]
EventProcessingEngine0=Combine
```

3. Create a new configuration section for the task, and set the following parameters:

`Type`	The ESP engine to use. Set this parameter to **`combine`**.
`Input0`	The first input track. This track must be an output track produced by another task.
`Input1`	The second input track. This track must be an output track produced by another task.
`MaxTimeInterval`	The maximum difference in time (in milliseconds) between a record in the first track to a record in the second, for the records to be considered as related. If you are processing images or documents this parameter is ignored.
`MinTimeInterval`	(Optional) The minimum difference in time (in milliseconds) between a record in the first track to a record in the second, for the records to be considered as related. The default value is the negative of the `MaxTimeInterval` value, meaning that the event in the second track can occur before the event in the first track (up to the specified number of milliseconds). If you are processing images or documents this parameter is ignored.

   For more details about these parameters, including the values that they accept, refer to the *Media Server Reference*.

4. (Optional) To add custom logic that discards pairs of records unless they meet additional conditions, set the `LuaScript` parameter so that Media Server runs a Lua script to filter the results. For information about writing the script, see "Write a Lua Script for an ESP Engine" on the next page.

`LuaScript`	The path and file name of a Lua script to run.

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following example runs face detection on an image file and blurs all of the faces that appear in the image. The *combine* task is used to combine the regions identified by face detection with the original image record produced by the ingest engine.

```
[Ingest]
IngestEngine=Image

[Image]
Type=image

[Analysis]
AnalysisEngine0=FaceDetect

[FaceDetect]
```

```
Type=FaceDetect
FaceDirection=any
Orientation=any

[EventProcessing]
EventProcessingEngine0=Combine

[Combine]
Type=combine
Input0=Image_1
Input1=FaceDetect.Result

[Transform]
TransformEngine0=Blur

[Blur]
Type=Blur
Input=Combine.Output

[Encoding]
EncodingEngine0=ToDisk

[ToDisk]
Type=ImageEncoder
ImageInput=Blur.Output
OutputPath=./_outputEncode/%token%.jpg
```

Write a Lua Script for an ESP Engine

All of the ESP engines, except for the "Or" engine, can run a Lua script to determine whether to include a record in the task's output track. Writing a Lua script allows you to specify more complex rules than you can specify using configuration parameters. For example, a Lua script might specify where in an image recognized text must appear. If text appears within this region, then depending on the engine type, the engine includes or excludes this record from its output track.

The Lua script must define a function with the name `pred`. This function takes one or two parameters (depending on the engine type) and must return `true` or `false`. Each parameter is a record object: this is a representation of the record being processed and has the same structure as the record XML.

The `pred` function is called once for each record (or pair of records) that the engine has to consider. The engine's response to the function depends on the engine type.

ESP Engine	Number of record parameters	Engine response if the function returns true
And	2	The record pair is included in the engine output track.
AndNot	2	The record in the first track is discarded. (Records from the

		second track are never included in the output anyway.)
AndNotThen	2	The record in the first track is discarded. (Records from the second track are never included in the output anyway.)
AndThen	2	The record pair is included in the engine output track.
Deduplicate	2	The second record is discarded.
Filter	1	The record is included in the engine output track.

When the `pred` function takes two parameters, each individual record may feature in many different pairs, so might be processed by the `pred` function any number of times. For example, for the AndNot or AndNotThen engine, a record in the first track might be passed to the function several times, being paired with different records from the second track. The record will only appear in the output track if `pred` returns `false` every time.

The ESP engine cannot modify the record objects passed to the `pred` function. Any effects of the function other than the return value are ignored.

To run a Lua script from an ESP engine, add the `LuaScript` parameter to the task configuration section and set it to the path of the script. For example, `LuaScript=./scripts/breakingnews.lua`.

Example Script

The following is an example script for the Filter ESP engine. The script filters records based on where text, read by an OCR task, appears in the image. The function returns `true` if text appears in the region between `x=100` and `x=300`, and the record is included in the output track. If the text region is outside these coordinates, the record is discarded.

```
function pred(rec)
   return rec.OCRResult.region.left > 100 and rec.OCRResult.region.right < 300
end
```

Part VI: Transform Data

This section describes how to transform the data produced by Media Server, so that you can customize how it is encoded or output to external systems.

Chapter 27: Crop Images

Many analysis tasks identify regions of interest in images or video frames. For example, face detection identifies the location of faces and number plate recognition identifies the location of number plates.

In some cases you might want to output the source image or video frame, cropped to show only the region of interest. This section describes how to crop images.

Crop Images

The **Crop** transformation task crops an image or video frame, to a region of interest that has been identified by another task. The task creates a new output track containing the cropped images. The track is named *taskName*.Output, where *taskName* is the name of the transformation task. The original input track is not modified.

The Crop transformation engine also translates any co-ordinates contained in records, so that their position is correct for the cropped image. For example, face detection outputs the locations of a person's eyes. The co-ordinates are modified so that the positions are correct for the cropped image.

Tip: To write the cropped images to disk, use the image encoder.

To crop images or video frames

1. Create a new configuration or open an existing configuration to send to Media Server with the process action. Alternatively, you can modify the Media Server configuration file (mediaserver.cfg).
2. In the [Transform] section, add a new transformation task by setting the TransformEngine*N* parameter. You can give the task any name, for example:

   ```
   [Transform]
   TransformEngine0=Crop
   ```

3. Create a new configuration section to contain the task settings and set the following parameters:

Type	The transformation engine to use. Set this parameter to **Crop**.
Input	The name of the image track that contains the images to crop, and supplies region data to use for cropping the images.

   For example:

   ```
   [Crop]
   Type=Crop
   Input=FaceRecognition.ResultWithSource
   ```

   For more information about the parameters that you can use to configure this task, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 28: Blur Regions of Images

Many analysis tasks identify regions of interest in images and video frames. In some cases you might want to blur these regions before encoding the images. For example, you can produce images where detected faces are blurred and are therefore unrecognizable.

Blur Images

The **Blur** transformation task blurs regions of an image or video frame. The task blurs any regions (including rectangles, polygons, and faces) that are present in the input records. It creates a new output track which contains the blurred images. This track is named *taskName*.Output, where *taskName* is the name of the transformation task. The original input track is not modified.

Tip: To write the blurred images to disk, use the image encoder.

To blur regions of images and video frames

1. Create a new configuration or open an existing configuration to send to Media Server with the process action. Alternatively, you can modify the Media Server configuration file (mediaserver.cfg).
2. In the [Transform] section, add a new transformation task by setting the TransformEngine*N* parameter. You can give the task any name, for example:

   ```
   [Transform]
   TransformEngine0=Blur
   ```

3. Create a new configuration section to contain the task settings and set the following parameters:

Type	The transformation engine to use. Set this parameter to **Blur**.
Input	The name of the track that contains the images to blur, with region data. The track must supply records that contain both an image and at least one region. Any regions (including rectangles, polygons, and faces) present in an input record are blurred in the output.

   For example:

   ```
   [Blur]
   Type=Blur
   Input=FaceDetect.ResultWithSource
   ```

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example Configuration

The following is an example configuration that blurs faces in a source image and encodes the resulting image to disk.

```
// ======================== Ingest ========================
[Ingest]
IngestEngine=Image

[Image]
Type=Image

// ======================== Analysis ======================
[Analysis]
AnalysisEngine0=FaceDetect

[FaceDetect]
Type=FaceDetect

// ======================== Transform =====================
[Transform]
TransformEngine0=Blur

[Blur]
Type=Blur
Input=FaceDetect.ResultWithSource

// ======================== Encoding ======================
[Encoding]
EncodingEngine0=ToDisk

[ToDisk]
Type=ImageEncoder
ImageInput=Blur.Output
OutputPath=./_outputEncode/%token%/blurred_faces.jpg

// ======================== Output ========================
[Output]
OutputEngine0 = ACI

[ACI]
Type=Response
Input=FaceDetect.Result
```

Chapter 29: Resize Images

Many processing tasks produce images. When Media Server ingests video, it decodes the video into individual frames. Analysis tasks can also produce images, for example keyframe extraction, face detection, and number plate recognition produce images of keyframes, faces, and number plates, respectively.

You might want to resize these images before encoding them. For example, you might extract keyframes for use in a web application, so that your users can navigate through a video file. In this case you might prefer to have thumbnail-size images rather than the high-definition video frames.

This section describes how to resize images in Media Server metadata tracks.

Resize Images

You can use a **scale** transformation task to duplicate an existing track and resize the images in the new track (the input track is not modified). The new output track is named *taskName*.Output, where *taskName* is the name of the transformation task.

The scale transformation engine also scales metadata that refers to the position of an object in the video frame, so that the positions remain correct after the image has been scaled. For example, when you resize images of faces, the metadata that describes the bounding box surrounding the face is also scaled.

To resize images

1. Create a new configuration or open an existing configuration to send to Media Server with the process action. Alternatively, you can modify the Media Server configuration file (mediaserver.cfg).
2. In the [Transform] section, add a new transformation task by setting the TransformEngine*N* parameter. You can give the task any name, for example:

   ```
   [Transform]
   TransformEngine0=ScaleKeyframes
   ```

3. Create a new configuration section to contain the task settings and set the following parameters:

Type	The transformation engine to use. Set this parameter to **Scale**.
Input	The name of the image track to process.
ImageSize	The output image size in pixels (width followed by height). If you specify one dimension and set the other to **0** (zero), Media Server preserves the aspect ratio of the original image.

   For example:

```
[ScaleKeyFrames]
Type=Scale
Input=Keyframe.ResultWithSource
ImageSize=300,0
```

For more information about the parameters that you can use to configure this task, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 30: Change the Format of Images

Many processing tasks produce images. When Media Server ingests video, it decodes the video into individual frames. Analysis tasks can also produce images, for example keyframe extraction, face detection, and number plate recognition produce images of keyframes, faces, and number plates, respectively.

You might want to change the format of these images before sending them to an output task (output tasks can output base-64 encoded image data).

Change the Format of Images

The **ImageFormat** transformation task transforms images in a track into another format. The new output track is named *taskName*.Output, where *taskName* is the name of the transformation task (the input track is not modified).

Tip: You can use the ImageFormat transformation engine to change the format of images before sending them to an output engine. When writing image files to disk through the image encoder, use the image encoder Format configuration parameter to specify the image format.

To change the format of images

1. Create a new configuration or open an existing configuration to send to Media Server with the process action. Alternatively, you can modify the Media Server configuration file (mediaserver.cfg).
2. In the [Transform] section, add a new transformation task by setting the TransformEngine*N* parameter. You can give the task any name, for example:

   ```
   [Transform]
   TransformEngine0=KeyframesFormat
   ```

3. Create a new configuration section to contain the task settings and set the following parameters:

Type	The transformation engine to use. Set this parameter to **ImageFormat**.
Input	The name of the image track to process.
Format	The output format for the images in the new track.

   For example:

   ```
   [KeyframesFormat]
   Type=ImageFormat
   Input=Keyframe.ResultWithSource
   Format=jpg
   ```

For more information about the parameters that you can use to configure this task, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Part VII: Output Data

Media Server can send the metadata it produces to many systems including IDOL Server, CFS, Vertica, and Milestone XProtect. Media Server can also write metadata to XML files on disk.

This section describes how to configure Media Server to output data.

Chapter 31: Introduction

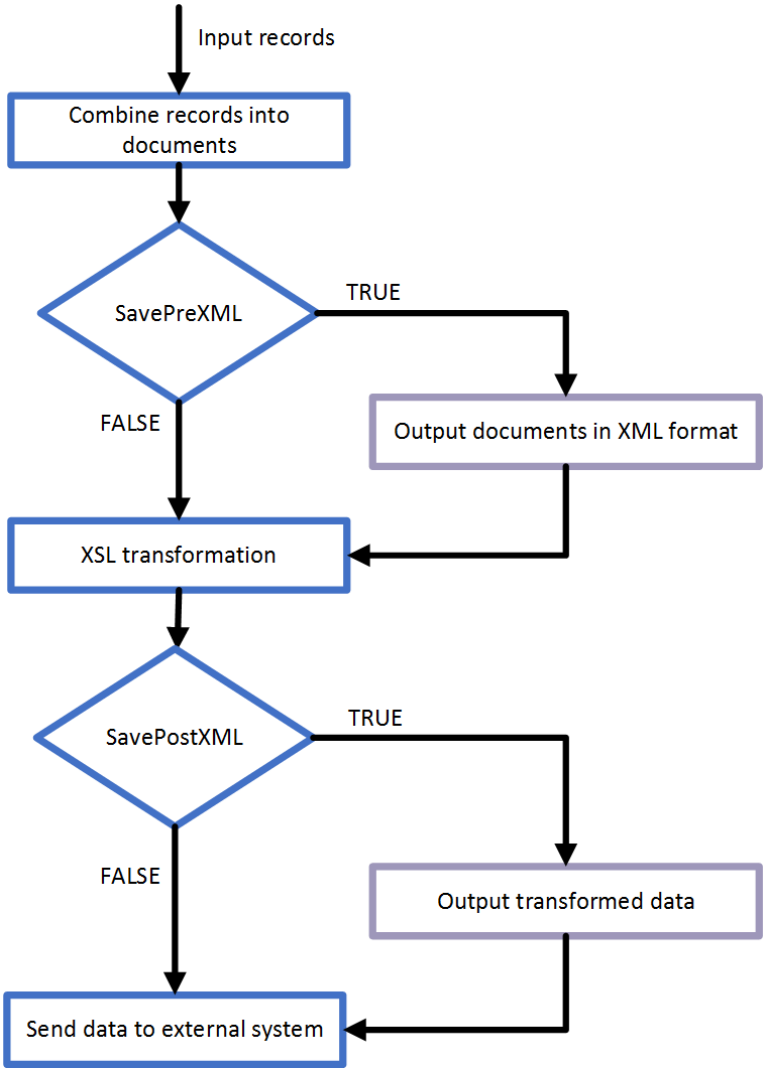
Media Server can output data in many formats, including XML and documents that you can send to CFS or index into IDOL Server.

Process Data

Media Server output tasks export the metadata produced by Media Server. This can include information about the ingested video, information about copies of the video that you create through encoding, and any information extracted from the video during analysis.

Tip: Output tasks do not output video. For information about saving video, see "Encode Video to a File or UDP Stream" on page 199

The following diagram shows the steps that occur when you configure Media Server to output data.



Select Input Records

An output task receives records that are produced by your ingest, analysis, encoding, transform, and ESP tasks. You can choose the information to output by setting the `Input` configuration parameter in the output task. If you do not set this parameter, the output task receives records from all tracks that are considered by default as 'output' tracks. For information about whether a track is considered by default to be an 'output' track, refer to *Media Server Reference*.

Combine Records into Documents

An output task receives individual records, but you might want to combine the information from many records and index that information as a single document. For example, if you are processing a news broadcast, you might want to index a document for each news story, rather than a document for each word spoken in the audio and a document for each recognized face. To do this, the Media Server must combine records representing recognized faces, speech-to-text results, recognized objects, and so on.

Most output engines have several indexing modes so that you can configure how to create documents. For more information about these indexing modes, see "Choose How to Output Data" below.

XSL Transformation

The output task performs an XSL transformation to convert the combined records into a format that is suitable for the destination repository.

Media Server is supplied with XSL templates to transform data into IDOL documents, Broadcast Monitoring assets, and other formats. You can modify the default templates to suit your needs. Set the `XSLTemplate` configuration parameter in the output task to specify the path of the XSL template to use.

If you want to customize the XSL template, or you need to troubleshoot a problem, set the configuration parameters `SavePreXML=TRUE`, `SavePostXML=TRUE`, and `XMLOutputPath` so that Media Server writes documents to disk before and after performing the transformation. Viewing the data before and after transformation might help you optimize your XSL template. In a production system, HPE recommends setting `SavePreXML` and `SavePostXML` to **`FALSE`**.

Send the Data to the External System

After performing the XSL transformation, Media Server sends the data to its destination.

Choose How to Output Data

Media Server analysis engines produce records, each of which describes something that happens in a video. A record might describe a recognized face, a scene change, or a word spoken in the audio. When you index data into some systems, such as IDOL Server or Broadcast Monitoring, you might want to combine the data from many records to produce documents that represent video segments or clips.

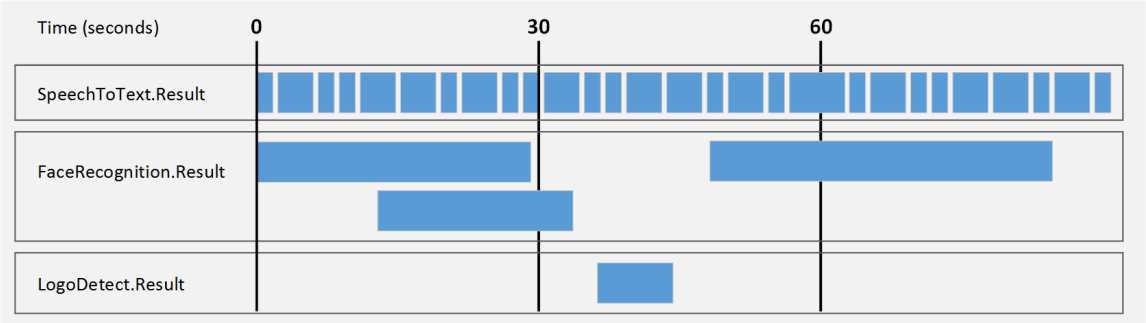
For example, the speech-to-text analysis engine produces a record for each word in the audio. If you are indexing data into a SQL database, you could insert a separate row into a database table for each word. If you are indexing data into IDOL Server, you might prefer to combine records so that each document contains the transcription from an entire news story or interview.

The following sections describe the indexing modes that you can choose when you configure Media Server to output data.

Single Record Mode

Single record mode creates documents that each represent a single record.

Consider the following records. In single record mode, Media Server creates a separate document for every record. No document contains more than one record.



This mode is suitable when you:

- send data to a database, where each record becomes a row in the database.
- need to index documents that represent discrete records, for example a recognized face or an ANPR result.
- need to use the metadata produced by Media Server in a front end application in real-time. In single record mode, Media Server outputs information about a record as soon as the record has finished. It does not need to hold records so that they can be combined with other related records.

Time Mode

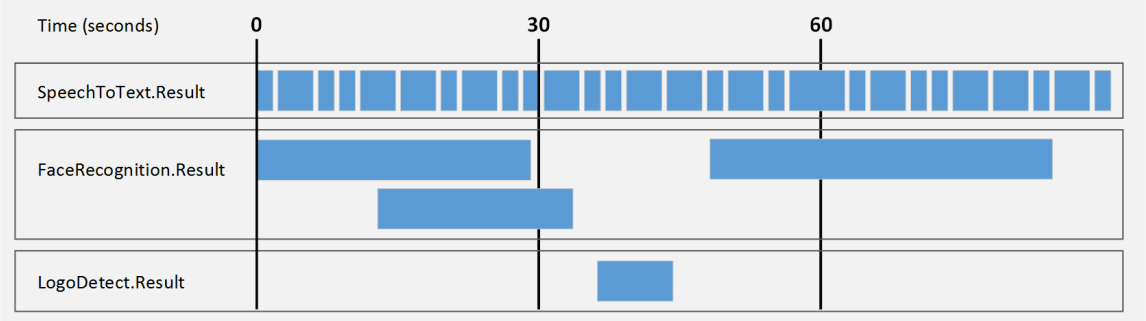
Time mode creates documents that represent a fixed amount of video content. A document contains all of the records that occurred during the time interval, or that overlap the start or end of the interval. You can define the duration of a document, for example 30 seconds. The duration is measured using video time, so you do not need to modify the duration if the ingest rate is slower or faster than normal (playback) speed.

Tip: Time-based output does not mean that Media Server outputs data at regular intervals. Media Server cannot output data for a video segment until all records in that segment have finished. As a result, Media Server might produce several documents, which represent consecutive time periods, at the same time.

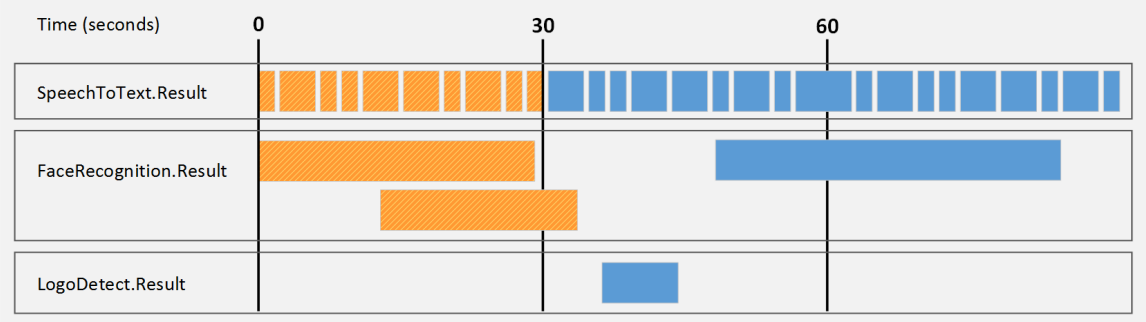
For example, if you run logo detection on a news broadcast, the news logo could be present on screen continuously for an hour. In that case, Media Server does not output any data until the logo disappears. Although Media Server creates documents that each represent 30 seconds of content, all of the documents are output at the end of the hour when the record describing the logo finishes.

In time mode, all records are output to at least one document. If a record spans more than one interval it is output to multiple documents.

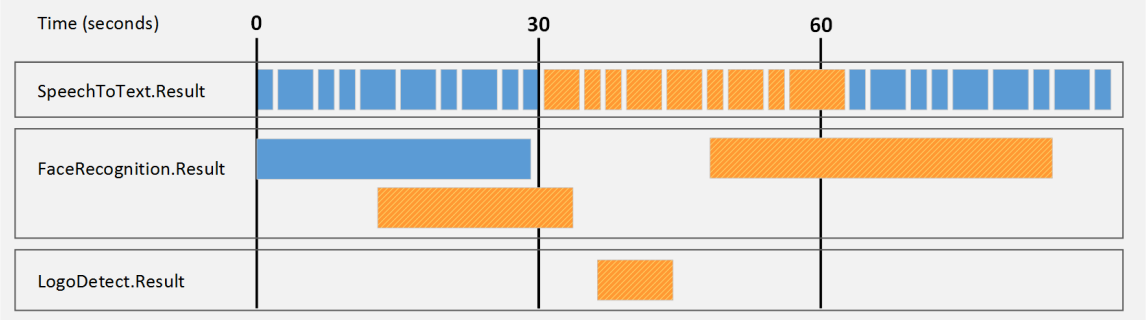
The following diagrams demonstrate how Media Server constructs documents in time mode. Consider the following records:



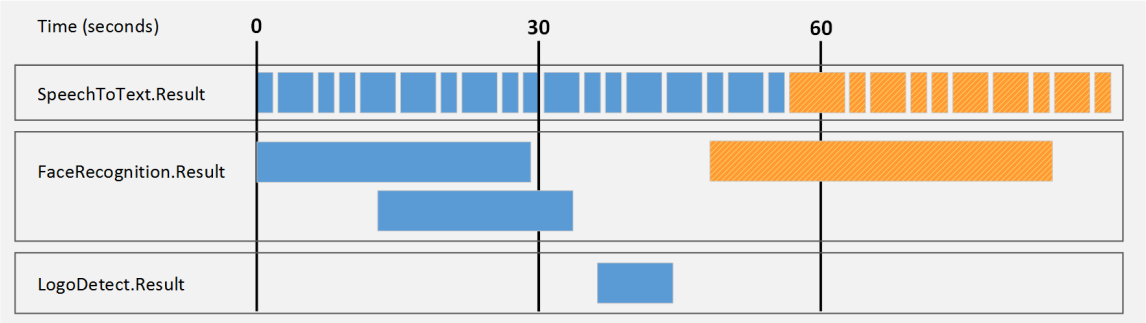
The first document contains the records related to the first interval (in this case, 30 seconds) of video content. Media Server does not output data at the 30-second point because a face recognition record has not ended (the face is still present in the video). Media Server can output data about the first interval as soon as this record ends:



The second document contains the records related to the second interval (in this case, 30 seconds) of video content. Notice that the second face recognition result is output in the second document as well, because it relates to both time intervals.



The third document contains the records related to the third interval (in this case, 30 seconds) of video content:



Time mode is simple to set up but documents might begin and end in the middle of a sentence or segment, rather than at meaningful point in the video.

Event Mode

Event mode helps to create documents that contain information about a single topic, which can provide a better experience for your users when they retrieve the content, and improves the performance of IDOL operations such as categorization.

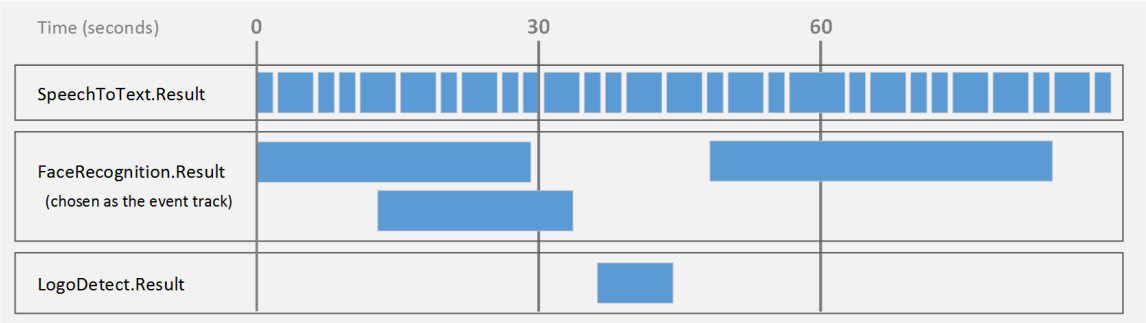
In event mode, Media Server creates a document for each record in an *event track*. The document that is generated contains the event from the event track. Other records are included if they overlap with the event in the event track or if they occurred after the end of the previous event. Each document might represent a different amount of video content.

The event track can be any track that you choose, though often it will be a track that you create using event stream processing. The event track could contain a record whenever there is a scene change, or whenever there is a pause in speech. For example, if you are analyzing news content Media Server can start a new document whenever there is a pause in speech, which could indicate the start of a new story.

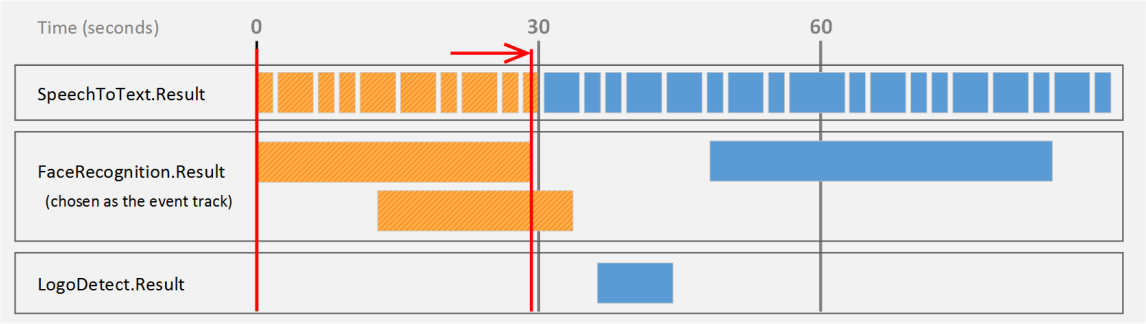
In event mode, all of the records in the selected tracks are output to at least one document. Compare this with "Bounded Event Mode" on page 248, in which some records can be omitted.

Note: Be aware that if you choose an event track that has overlapping records (for example the result track from a face recognition task), the resulting documents might contain more than one record from the event track, and some records will be output to multiple documents.

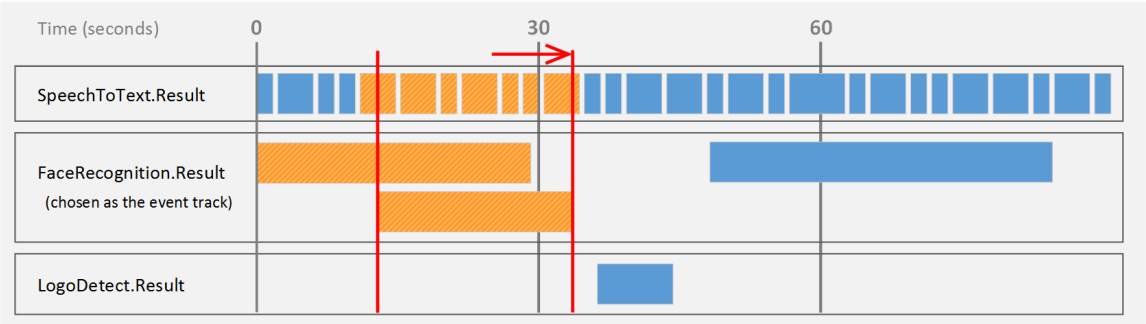
The following diagrams demonstrate how Media Server constructs documents in **event** mode. The `FaceRecognition.Result` track has been chosen as the event track. Consider the following records:



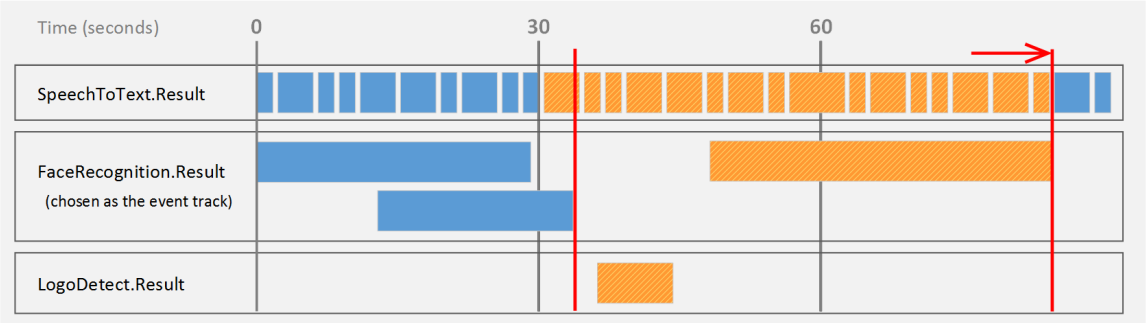
The first document contains the first record in the event track (the first event). Other records are included if they overlap with this event or if they occurred since the end of the previous event. In this case the previous event is the beginning of the video:



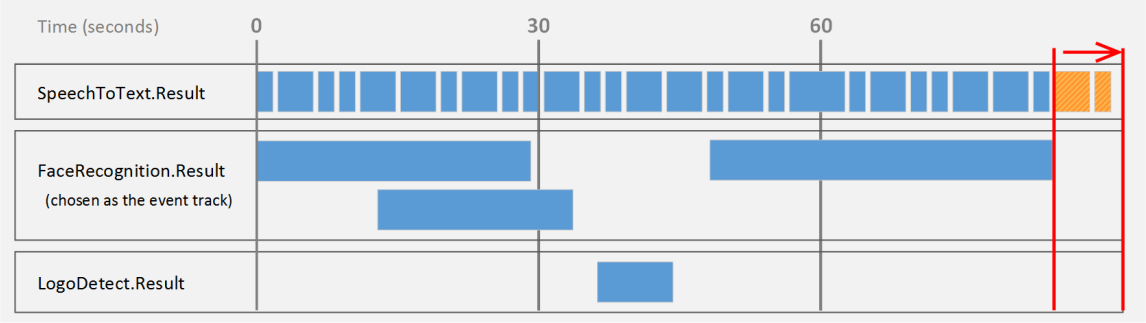
The second document contains the second record in the event track (the second event). Other records are included if they overlap with this event or if they occurred since the end of the previous event:



The third document contains the third record in the event track (the third event). Other records are included if they overlap with this event or if they occurred since the end of the previous event:



The end of the video is considered as an event, so in this case a final document is produced containing the final records in the speech-to-text result track:



Bounded Event Mode

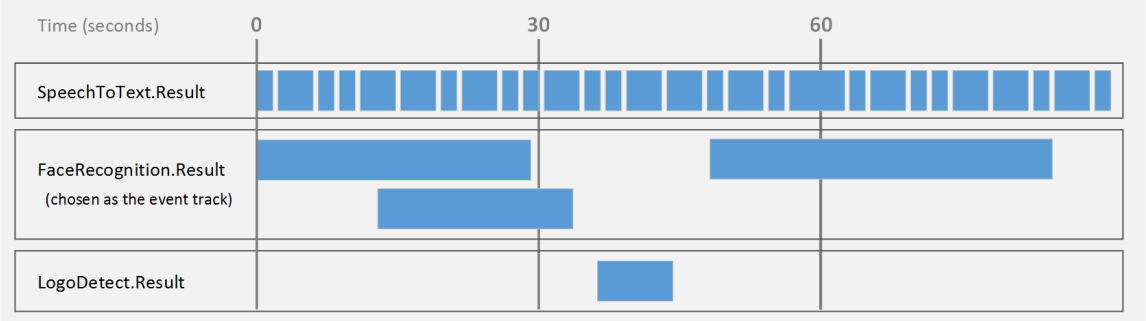
Bounded event mode, like event mode, helps to create documents that contain information about a single topic. Media Server creates a document for each record in an *event track*. The document that is generated contains the event from the event track. However, unlike event mode, other records are included only if they overlap with the event in the event track.

Note: In bounded event mode, it is possible that some records are not output to documents.

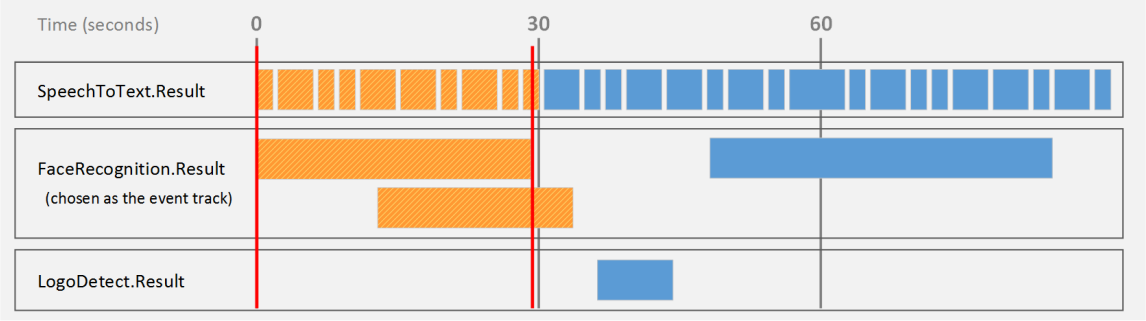
The event track can be any track that you choose, though often it will be a track that you create using event stream processing. You could use speaker identification results as your event track, so that a document is created for each speaker detected in the audio.

Each document might represent a different amount of video content.

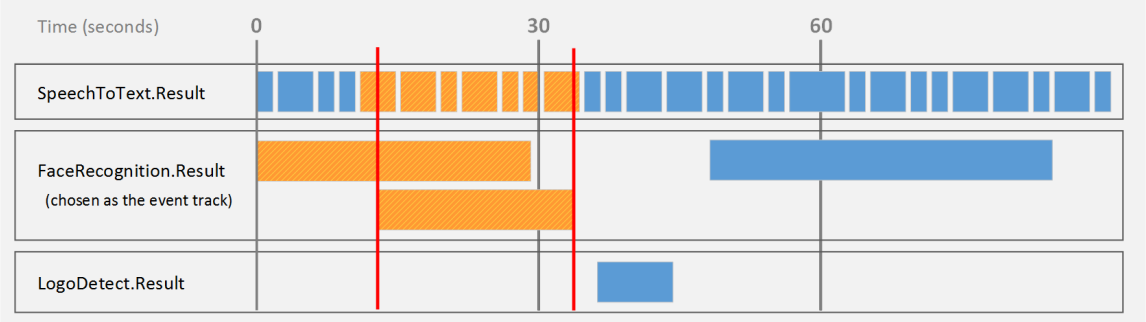
The following diagrams demonstrate how Media Server constructs documents in **bounded event** mode. The `FaceRecognition.Result` track has been chosen as the event track. Consider the following records:



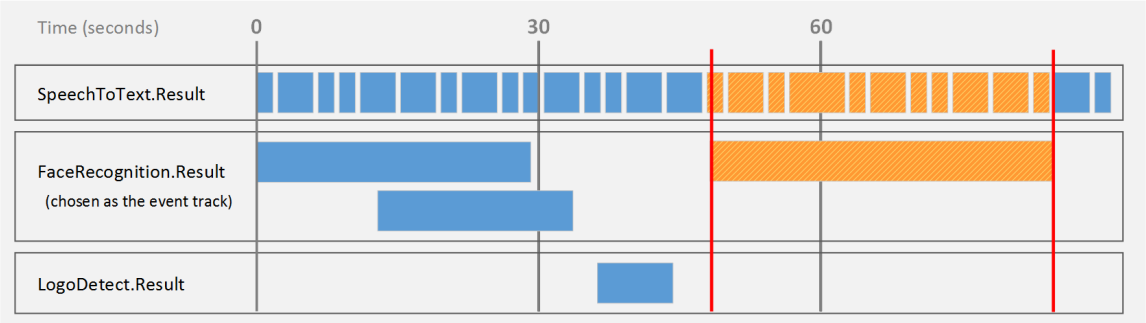
The first document contains the first record that occurs in the event track, and all of the records that occur at the same time:



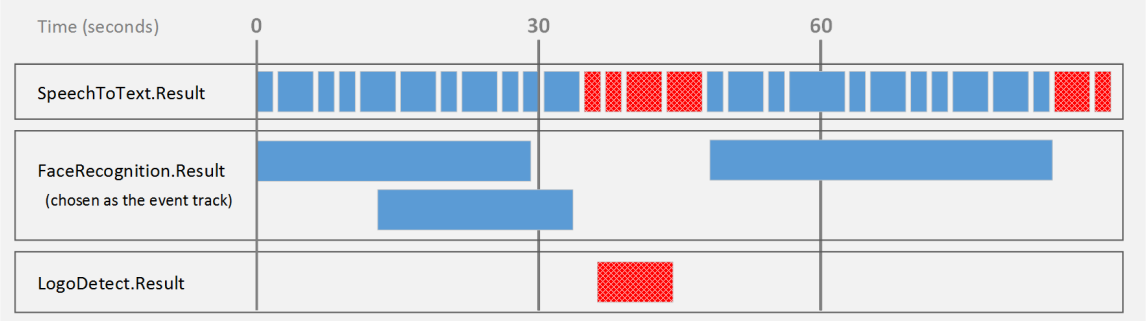
The second document contains the second record that occurs in the event track, and those records that overlap with it. Notice that if records in the event track overlap, the output document can contain more than one record from the event track:



The next document contains the next record from the event track, and the records that overlap with it:



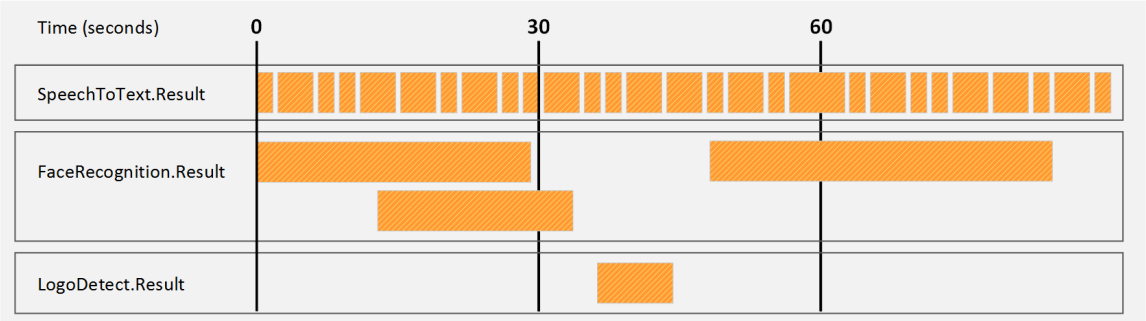
In bounded event mode, the records shown here are not output to any document, because they do not overlap with a record in the event track:



At End Mode

At End mode creates documents that each represent an asset such as a PDF file, image file, or video file. Media Server creates a maximum of one document for each `process` action. The document contains all of the records that were generated when Media Server processed that asset.

For example, if you process a video file, all of the records are output to the same document:



This mode is suitable when you are processing image files, because all of the information extracted from the image is added to a single document. You can also use this mode if you are processing video assets but want all of the information about a video file to be indexed as a single document.

At End mode is not suitable if you are processing video streams for broadcast monitoring or surveillance purposes, because Media Server does not output information until processing is complete.

Information about events would not be output until hours, days, or weeks after the events had occurred, because in these scenarios `process` actions are expected to run for a long time.

Page Mode

Page mode creates documents that represent a page of a processed image or document.

You can use this mode only when ingesting images and documents.

Some image file formats (for example TIFF) support multiple pages, and some document formats (such as Adobe PDF) provide page numbers for embedded text and images. If you are processing multi-page images or documents, you can use page mode to create separate documents for each page.

Chapter 32: ACI Response

This section describes how to configure Media Server to output data in the response to the `process` action.

Introduction

Media Server does not output records to the `process` action response by default.

You can configure Media Server to do this so that another server, such as a Connector Framework Server, can send requests to Media Server for analysis and then retrieve the results from the action response.

Tip: HPE recommends that you do not write results to the action response in cases where Media Server produces a large amount of data. If you output data from tracks that contain a large amount of metadata, the ACI response could become extremely large. This might result in the system running out of resources, or external applications failing to retrieve the response.

Output Data to the Process Action Response

To write records to the action response

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Output]` section, add a new output task by setting the `OutputEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=Response
   ```

3. Create a configuration section for the task and set the following parameters:

`Type`	The output engine to use. Set this parameter to **`response`**.
`Input`	(Optional) A comma-separated list of the tracks that you want to output. Specify one or more tracks. If you do not set this parameter, the engine outputs all tracks that are configured by default as 'output' tracks. For information about whether a track is considered an 'output' track, refer to documentation for your analysis and encoding engines in the *Media Server Reference*.

   For more information about these parameters, refer to the *Media Server Reference*.

4. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following is an example configuration to output data to the `process` action response.

```
[Response]
Type=response
Input=OCR.Result,Keyframe.Result
```

Chapter 33: Files on Disk

This section describes how to configure Media Server to output data to files on disk.

Output Data to Files

Media Server can output records to files, so that you can index the information into any system that accepts data in a format such as XML.

To write records to files, use the XML output engine. The default format is XML but you can configure the engine to apply an XSL transformation to the output, to transform it into another format such as HTML.

To write records to files

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Output]` section, add a new output task by setting the `OutputEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=XmlWriter
   ```

3. Below the `[Output]` section, create a new configuration section by typing the task name inside square brackets. For example:

   ```
   [XmlWriter]
   ```

4. In the new section, set the following parameters:

`Type`	The output engine to use. Set this parameter to **`XML`**.
`Input`	(Optional) A comma-separated list of the tracks that you want to output. Specify one or more tracks. If you do not set this parameter, the engine outputs all tracks that are configured by default as 'output' tracks. For information about whether a track is considered an 'output' track, refer to documentation for your analysis and encoding engines in the *Media Server Reference*.
`XmlOutputPath`	The path and file name of the XML files to create.
`XSLTemplate`	(Optional) The path to the XSL template to use to transform the output into the desired format. If you do not set this parameter, the output is not transformed.

   For more information about the parameters that you can use to customize an XML output task, refer to the *Media Server Reference*.

5. Configure how to combine records into XML files. For information about how you can combine records, see "Choose How to Output Data" on page 243.
   - To output data in single record mode, set `Mode=SingleRecord`.
   - To output data in time mode, set `Mode=Time` and use the `OutputInterval` parameter to specify the amount of video content represented by each document.
   - To output data in event mode, set `Mode=Event` and use the `EventTrack` parameter to specify the event track.
   - To output data in bounded event mode, set `Mode=BoundedEvent` and use the `EventTrack` parameter to specify the event track.
   - To output data in at-end mode, set `Mode=AtEnd`.
   - To output data in page mode, set `Mode=Page`.
6. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following is an example configuration to output data using the XML output engine.

```
[XmlWriter]
Type=xml
XMLOutputPath=./output/html/%segment.type%_
results_%timestamp%_%segment.sequence%.html
XSLTemplate=./xsl/tohtml.xsl
Mode=Time
OutputInterval=30
```

Chapter 34: Connector Framework Server

This section describes how to configure Media Server to send IDOL documents to Connector Framework Server (CFS).

Send Documents to Connector Framework Server

Media Server includes an output engine for indexing documents into IDOL Server. However, if you want to manipulate and enrich documents before they are indexed into IDOL you can send the documents to a Connector Framework Server (CFS) instead.

For example, if you have used speech-to-text analysis to convert the words spoken in a video into text, you might want to run Eduction on the text to extract the names of people or places and add these to the document metadata.

To send documents to CFS, use the CFS output engine. Media Server includes an XSL template that you can use to transform the metadata produced by Media Server into documents.

For more information about using CFS, refer to the *Connector Framework Server Administration Guide*.

To send documents to CFS

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. Add a new output task by setting the `OutputEngine`*N* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=CFS
   ```

3. Create a section to configure the task by typing the task name inside square brackets. For example:

   ```
   [CFS]
   ```

4. In the new section, set the following parameters:

`Type`	The output engine to use. Set this parameter to **`CFS`**.
`CfsHost`	The host name or IP address of your CFS.
`CfsPort`	The ACI port of your CFS.
`XslTemplate`	The path to the XSL template to use to transform records into documents. Media Server includes an XSL template for sending documents to CFS. This template is named `toCFS.xsl` and is in the Media Server installation directory.

`Input`	(Optional) A comma-separated list of the tracks that you want to output. Specify one or more tracks. If you do not set this parameter, the engine outputs all tracks that are configured by default as 'output' tracks. For information about whether a track is considered an 'output' track, refer to documentation for your analysis, encoding, and ESP engines in the *Media Server Reference*.

5. Configure how to combine records into documents. For information about how you can combine records, see "Choose How to Output Data" on page 243.
   - To output data in single record mode, set `Mode=SingleRecord`.
   - To output data in time mode, set `Mode=Time` and use the `OutputInterval` parameter to specify the amount of video content represented by each document.
   - To output data in event mode, set `Mode=Event` and use the `EventTrack` parameter to specify the event track.
   - To output data in bounded event mode, set `Mode=BoundedEvent` and use the `EventTrack` parameter to specify the event track.
   - To output data in at-end mode, set `Mode=AtEnd`.
   - To output data in page mode, set `Mode=Page`.
6. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following is an example configuration to send documents to CFS.

```
[CFS]
Type=CFS
CfsHost=localhost
CfsPort=7000
XSLTemplate=./xsl/toCFS.xsl
Mode=time
OutputInterval=20
```

Chapter 35: IDOL Server

This section describes how to configure Media Server to index data into IDOL Server.

Set up an IDOL Output Task

Media Server's IDOL output engine transforms metadata produced by Media Server into IDOL documents and indexes the documents into an IDOL Server.

The IDOL output engine uses an XSL template to transform records into IDX files. To modify the structure of the IDX file, modify the XSL template. For more information about indexing content into IDOL Server, refer to the *IDOL Server Administration Guide*.

To index information into IDOL Server

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Output]` section, add a new output task by setting the `OutputEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=IDOL
   ```

3. Below the `[Output]` section, create a configuration section for the engine by typing the task name inside square brackets. For example:

   ```
   [IDOL]
   ```

4. In the new section, set the following parameters:

`Type`	The output engine to use. Set this parameter to **`IDOL`**.
`Input`	(Optional) A comma-separated list of the tracks that you want to output. Specify one or more tracks. If you do not set this parameter, the engine outputs all tracks that are configured by default as 'output' tracks. For information about whether a track is considered an 'output' track, refer to documentation for your analysis, encoding, and ESP engines in the *Media Server Reference*.
`IdolHost`	(Optional) The host name or IP address of the IDOL Server. This overrides the IDOL Server host specified by the `IdolServer` parameter in the `[Resources]` section, if it has been set.
`IdolPort`	(Optional) The ACI port of the IDOL Server (by default, `9000`). This overrides the port specified by the `IdolServer` parameter in the `[Resources]` section, if it has been set.

`IdolDB`	(Optional) The IDOL database to index documents into. This overrides the database specified by the `IdolServer` parameter in the `[Resources]` section, if it has been set. If you do not set this parameter, documents are indexed into the database specified by their `DREDBNAME` metadata field. You can modify your XSL template to create this field. If a document does not specify a database it is indexed into the default database specified by the `DefaultDatabase` parameter, in the `[Databases]` section of the IDOL Server configuration file.
`IDOLParams`	(Optional) Additional IDOL index action parameters. The IDOL output engine sends the `DREADDDATA` action to IDOL Server, which instructs the server to index the data contained in the request. You can send additional parameters with this action. For information about the available index action parameters, refer to the *IDOL Server Reference*.
`XSLTemplate`	The path to the XSL template to use to transform records into documents in IDX format. You can modify the default XSL template as required - for example to produce XML rather than IDX files.
`SavePostXML`	(Optional) Specifies whether to save IDX files produced by the IDOL output engine. If this parameter is set to **`true`**, you must also set the `XMLOutputPath` parameter.
`SavePreXML`	(Optional) Specifies whether to save records received by the IDOL output engine. If this parameter is set to **`true`**, you must also set the `XMLOutputPath` parameter.
`XMLOutputPath`	(Optional) The path and file name of the file to save pre-XML and post-XML output to.

5. Configure how to combine records into documents. For information about how you can combine records, see "Choose How to Output Data" on page 243.
   - To output data in single record mode, set `Mode=SingleRecord`.
   - To output data in time mode, set `Mode=Time` and use the `OutputInterval` parameter to specify the amount of video content represented by each document.
   - To output data in event mode, set `Mode=Event` and use the `EventTrack` parameter to specify the event track.
   - To output data in bounded event mode, set `Mode=BoundedEvent` and use the `EventTrack` parameter to specify the event track.
   - To output data in at-end mode, set `Mode=AtEnd`.
   - To output data in page mode, set `Mode=Page`.
6. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following is an example configuration to output data using the IDOL output engine.

```
[IDOLOutput]
Type=IDOL
IdolHost=localhost
IdolPort=9000
IdolDB=BroadcastVideo
Mode=Time
OutputInterval=30
XslTemplate=./xsl/toIDX.xsl
SavePreXML=true
SavePostXML=true
XMLOutputPath=./Output/%segment.type%_%segment.sequence%_%segment.timestamp%.xml
```

Chapter 36: Vertica Database

This section describes how to configure Media Server to insert data into a Vertica database.

Insert Data into a Vertica Database

To insert records into a Vertica database, use the Vertica output engine.

The Vertica output engine uses an XSL template to transform the XML produced by Media Server into a format, such as a CSV file, that can be inserted into the database. It then connects to the database using ODBC and inserts the information using a `COPY` query:

```
COPY <table>
FROM LOCAL '<local_file>'
DELIMITER '<delimiter>'
ENCLOSED BY '<quote>'
ESCAPE AS '<escape>'
```

where:

`<table>` is the Vertica database table to copy data into. This is read from the `TrackMapping` configuration parameter.

`<delimiter>`, `<quote>`, and `<escape>` are replaced by values from the corresponding configuration parameters.

To insert records into a Vertica database

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Output]` section, add a new output task by setting the `OutputEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=VerticaOutput
   ```

3. Below the `[Output]` section, create a configuration section for the engine by typing the task name inside square brackets. For example:

   ```
   [VerticaOutput]
   ```

4. In the new section, set the following parameters:

`Type`	The output engine to use. Set this parameter to **`vertica`**.
`TrackMapping`	The tracks that you want to output, mapped to Vertica database tables.

`OdbcConnectionString`	The ODBC connection string to use to connect to the database. For information about how to connect to a Vertica database, refer to the Vertica documentation.
`OdbcDriverManager`	(Required only on UNIX platforms) The path of the ODBC driver manager to use.
`XMLOutputPath`	The path to the directory to use for temporary files and saved output.
`XSLTemplate`	The XSL template to use to transform records from analysis engines to a format that can be inserted into the database (such as a CSV file).
`OutputInterval`	(Optional) The interval, in seconds, between inserting batches of records into the database. The default interval is 60 seconds.

For example:

```
[VerticaOutput]
Type=vertica
TrackMapping0=FaceRecog.Result : face_recognition
TrackMapping1=Ocr.Result : ocr
OdbcConnectionString=DSN=mydb
OdbcDriverManager=libodbc.so
XMLOutputPath=./tmp
XSLTemplate=./xsl/toCSV.xsl
OutputInterval=120
```

For more information about the parameters that you can set to configure a Vertica output task, refer to the *Media Server Reference*.

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 37: ODBC Database

This section describes how to configure Media Server to insert data into a database that can be accessed through ODBC.

Insert Records into a Database

You can use the ODBC output engine to insert information into a database through ODBC.

The ODBC output engine uses an XSL template to extract data from Media Server records. Your XSL template must produce a file which contains a list of transactions to run. Apart from the last, each transaction must end with the delimiter `==END==`, followed by a new line character.

Each transaction can contain one or more queries, separated by new line characters, and each query should have the following format:

- A statement to run.
- A list of alternating, tab-separated, column types and column values.

The following is an example post-XML file that contains five transactions, each with a single query.

```
insert into segment(uuid, sessionToken, startTime, endTime) VALUES (?, ?, ?, ?);
string 67589a12-d7fe-44c6-915f-ad36b20e39da string
MTYuMjguOTQuMTMxOjE0MDAwOlBST0NFU1M6MTQ2MDM3MzY4OTMxNzAxNDMyODM0MDcw bigint
1460373884855305 bigint 1460373899855305

==END==

insert into speech(segmentId, startTime, duration, text, confidence) VALUES (?, ?,
?, ?, ?); string 67589a12-d7fe-44c6-915f-ad36b20e39da bigint 1460373898073305
bigint 450000 string satellite double 0

==END==

insert into speech(segmentId, startTime, duration, text, confidence) VALUES (?, ?,
?, ?, ?); string 67589a12-d7fe-44c6-915f-ad36b20e39da bigint 1460373898523305
bigint 370000 string images double 0

==END==

insert into speech(segmentId, startTime, duration, text, confidence) VALUES (?, ?,
?, ?, ?); string 67589a12-d7fe-44c6-915f-ad36b20e39da bigint 1460373898893305
bigint 90000 string are double 0
```

```
==END==

insert into speech(segmentId, startTime, duration, text, confidence) VALUES (?, ?,
?, ?, ?); string 67589a12-d7fe-44c6-915f-ad36b20e39da bigint 1460373898983305
bigint 420000 string released double 0
```

Tip: Media Server only executes valid queries. A valid query must have a statement to run, must have the same number of column types and column values, and must not attempt to insert data into more than 100 columns. If a query results in an error, the entire transaction is rolled back.

To see an example XSL transformation that converts information into the correct, format, see `./xsl/toODBC.xsl` in the Media Server installation folder.

Before You Begin

If you are running Media Server on Linux, ODBC connector drivers for your database might be included in the operating system distribution, or be available from a package manager. It is likely that later versions of the connector driver will be available for download directly from the database provider. The later drivers might contain stability and performance improvements. If you experience issues using the ODBC output engine, HPE recommends downloading the latest ODBC connector drivers for your database as the first step in the troubleshooting process.

If you have configured the ODBC output engine to output data into a table or column that has an extended Unicode character (a character that is not included in the ASCII character set) in its name, then you must use a Unicode ODBC driver. These drivers are often identified by a "w" being appended to the driver name.

Configure the Output Task

To configure an output task to insert information into a database, follow these steps.

To insert information into a database

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Output]` section, add a new output task by setting the `OutputEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=MyDatabase
   ```

3. Below the `[Output]` section, create a configuration section for the engine by typing the task name inside square brackets. For example:

   ```
   [MyDatabase]
   ```

4. In the new section, set the following parameters:

`Type`	The output engine to use. Set this parameter to **`ODBC`**.

`Input`	(Optional) A comma-separated list of the tracks that contain information you want to output. Specify one or more tracks. If you do not set this parameter, the engine outputs all tracks that are configured by default as 'output' tracks. For information about whether a track is considered an 'output' track, refer to documentation for your analysis, encoding, and ESP engines in the *Media Server Reference*.
`OdbcConnectionString`	The ODBC connection string to use to connect to the database.
`OdbcDriverManager`	(Linux only) The path of the ODBC driver manager shared library. This parameter is not required if you are running Media Server on Windows.
`XSLTemplate`	The path to the XSL template to use to extract values from records and construct a list of queries to run against the database.

5. Configure how to combine records into SQL queries. For more information about how you can combine records, see "Choose How to Output Data" on page 243.
   - To output data in single record mode, set `Mode=SingleRecord`.
   - To output data in time mode, set `Mode=Time` and use the `OutputInterval` parameter to specify the amount of video content represented by each document.
   - To output data in event mode, set `Mode=Event` and use the `EventTrack` parameter to specify the event track.
   - To output data in bounded event mode, set `Mode=BoundedEvent` and use the `EventTrack` parameter to specify the event track.
   - To output data in at-end mode, set `Mode=AtEnd`.
   - To output data in page mode, set `Mode=Page`.
6. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following is an example configuration to output data using the ODBC output engine.

```
[MyDatabase]
Type=odbc
OdbcConnectionString=DSN=MyDSN;
Mode=singlerecord
XslTemplate=./xsl/toODBC.xsl
SavePreXML=true
SavePostXML=true
XMLOutputPath=./output/odbc/%segment.type%_%segment.sequence%.xml
```

Example Configuration

Media Server includes an example SQL script, configuration file, and XSL template that demonstrate how to output data to a database through ODBC.

The example configuration runs OCR and speech-to-text on a video file or stream. It also encodes the video to disk in 30 second segments.

Tip: Speech-to-text requires an IDOL Speech Server. If you want to run a task from the example configuration, define the location of the Speech Server in your Media Server configuration file, by setting the `SpeechToTextServers` parameter in the `[Resources]` section.

The example includes the following files:

- `configurations/broadcast_schema.sql` is an example SQL script that creates tables in your database to store information about the processed video, and information extracted by OCR and speech-to-text. If you want to run a task from the example configuration, use this script to create the required tables in your database.
- `configurations/broadcastODBC.cfg` is an example configuration file that outputs data through ODBC. Before starting a `process` action with this configuration, make the following changes:
  - In the `[ODBC]` section, set the configuration parameter `ODBCConnectionString` to the connection string to use to connect to your database.
  - Unless you have defined the location of your Speech Server in the Media Server configuration file, find the `[SpeechToText]` section and set the `SpeechToTextServers` parameter to the host name and ACI port of your Speech Server.
- `xsl/toODBC.xsl` is an XSL template that takes the data produced by Media Server and creates transactions for inserting the data into the database.

Insert Image Data into a Database

Some of the tracks that are produced by Media Server engines contain binary data (images). Usually Media Server output engines output this data in base-64 encoded form. However, when the ODBC output engine creates and processes pre- and post-XML, it replaces the image data with a GUID. This prevents the XML becoming excessively large and increases the speed of the XSL transformation. When the engine inserts information into your database, the GUID is replaced by the actual binary data.

If you use the ODBC output engine to write images, such as keyframes, into a database, you can therefore insert the images into a column that accepts binary object (BLOB) data.

Troubleshooting

Information is not inserted into the database.

If information is not inserted into the database, ensure that Media Server can connect to the database. If Media Server cannot send information to the database, the information is saved to an

XML file named as follows:

- If either `SavePreXml` or `SavePostXml` is set to `true`, Media Server saves the information to the directory specified by `XmlOutputPath`.
- If `SavePreXml` and `SavePostXml` are both set to `false`, the information is saved to `./failed/`*`sessionToken`*`/`*`taskName`*`/failed_segmentNumber.xml`, where *`sessionToken`* is the asynchronous action token and *`taskName`* is the name of the ODBC output task.

If the connection to the database is lost and re-established, Media Server continues inserting information, but does not insert the records that were saved to disk. You must insert these into the database manually.

Chapter 38: HTTP POST

This section describes how to configure Media Server to output data by sending the data in the body of HTTP POST requests.

Send Information over HTTP

Media Server includes an output engine for sending data to a server through an HTTP POST request. The body of the request contains the information produced by the XSL transformation.

To send information over HTTP

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. Add a new output task by setting the `OutputEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=HTTPpost
   ```

3. Create a configuration section for the engine by typing the task name inside square brackets. For example:

   ```
   [HTTPpost]
   ```

4. In the new section, set the following parameters:

`Type`	The output engine to use. Set this parameter to **`httppost`**.
`DestinationURL`	The URL to send the information to. You can include macros in the URL.
`XSLTemplate`	The path to the XSL template to use to transform records and produce the body of the request.
`Input`	(Optional) A comma-separated list of the tracks that you want to include in the output. Specify one or more tracks. If you do not set this parameter, the engine includes all tracks that are preset as 'output' tracks. For information about whether a track is considered an 'output' track, refer to documentation for your analysis, encoding, and ESP engines in the *Media Server Reference*.

5. Configure how to combine records into HTTP requests. For information about how you can combine records, see "Choose How to Output Data" on page 243.

- To output data in single record mode, set `Mode=SingleRecord`.
- To output data in time mode, set `Mode=Time` and use the `OutputInterval` parameter to specify the amount of video content represented by each document.
- To output data in event mode, set `Mode=Event` and use the `EventTrack` parameter to specify the event track.
- To output data in bounded event mode, set `Mode=BoundedEvent` and use the `EventTrack` parameter to specify the event track.
- To output data in at-end mode, set `Mode=AtEnd`.
- To output data in page mode, set `Mode=Page`.

6. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Example

The following is an example configuration to send documents to a CFS over HTTP.

```
[HTTPpost]
Type=httppost
DestinationURL=http://localhost:7000/action=ingest
XSLTemplate=./xsl/toCFS.xsl
Mode=event
EventTrack=NewsSegment.Result
```

Chapter 39: Broadcast Monitoring

This section describes how to configure Media Server to index data into Broadcast Monitoring.

Index Records into Broadcast Monitoring

The Broadcast Monitoring output engine transforms information produced by Media Server and indexes it into Broadcast Monitoring.

Media Server segments data into documents, which each represent a fixed amount of video content. You can specify how many documents to index into each asset.

To index information into Broadcast Monitoring

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Output]` section, add a new output task by setting the `OutputEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=BroadcastMonitoring
   ```

3. Below the `[Output]` section, create a configuration section for the engine by typing the task name inside square brackets. For example:

   ```
   [BroadcastMonitoring]
   ```

4. In the new section, set the following parameters:

`Type`	The output engine to use. Set this parameter to **`BM`**.
`BroadcastMonitoringTrickleURL`	The Broadcast Monitoring URL to send data to. Setting this parameter overrides the setting specified in the `[Resources]` section, if `BroadcastMonitoringTrickleURL` has been set there.
`Input`	(Optional) A comma-separated list of the tracks that you want to output. Specify one or more tracks. If you do not set this parameter, the engine outputs all tracks that are configured by default as 'output' tracks. For information about whether a track is considered an 'output' track, refer to documentation for your analysis, encoding, and ESP engines in the *Media Server Reference*.
`OutputInterval`	The amount of video content represented by each output document, in seconds.

`IntervalsPerAsset`	The number of output documents to index into an asset in Broadcast Monitoring. After this number of intervals, Media Server starts a new asset. For example, if you set `OutputInterval=30` and `IntervalsPerAsset=10`, Media Server sends documents to Broadcast Monitoring that represent 30 seconds of video content. After 10 of these documents have been sent to Broadcast Monitoring and the asset represents 5 minutes of video, Media Server starts a new asset.
`XSLTemplate`	The path to the XSL template to use to transform the output so that it can be indexed into Broadcast Monitoring. HPE recommends that you use the XSL template supplied with Media Server, because Broadcast Monitoring requires the data in a specific format.
`LabelSetBroadcaster`	The name of the video broadcaster, to add to the "Broadcaster" field in the Broadcast Monitoring label set.
`LabelSetCompleted`	The value to use for the "Completed" field in the Broadcast Monitoring label set (**`true`**/**`false`**).
`LabelSetProgram`	The program name, to add to the "Program" field in the Broadcast Monitoring label set.
`DefaultProxyTrack`	The proxy track to output to Broadcast Monitoring as the default proxy track.
`ProxyTracks`	(Optional) A comma-separated list of additional proxy tracks to output to Broadcast Monitoring.
`SavePostXML`	(Optional) Specifies whether to save XML files produced by the Broadcast Monitoring engine. If this parameter is set to **`true`**, you must also set the `XMLOutputPath` parameter.
`SavePreXML`	(Optional) Specifies whether to save records received by the Broadcast Monitoring engine. If this parameter is set to **`true`**, you must also set the `XMLOutputPath` parameter.
`XMLOutputPath`	(Optional) The path and file name of the file to save pre-XML and post-XML output to.

5. (Optional) You can set further optional parameters. For information about the configuration parameters that you can use with a Broadcast Monitoring output task, refer to the *Media Server Reference*.
6. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Chapter 40: Milestone XProtect

This section describes how to configure Media Server to send data to a Milestone XProtect, a third-party video management system.

Introduction

Media Server's Milestone output engine sends data to Milestone XProtect Corporate and XProtect Enterprise surveillance systems.

The Milestone Smart Client displays the events detected by Media Server, by showing information overlaid on the video. For example, detected objects are identified by bounding polygons. The Smart Client also shows metadata produced by Media Server, such as a message, event type, and location.

Before You Begin

To use Media Server with a Milestone XProtect surveillance system, HPE recommends using:

- Milestone XProtect Enterprise 8.1a (96231)
- Milestone XProtect Corporate 2013 R2

Note: Other versions might work but have not been tested.

Configure Media Server

To send data to Milestone XProtect, follow these steps.

To send data to Milestone XProtect

1. Create a new configuration or open an existing configuration to send to Media Server with the `process` action. Alternatively, you can modify the Media Server configuration file (`mediaserver.cfg`).
2. In the `[Output]` section, add a new output task by setting the `OutputEngine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Output]
   OutputEngine0=XProtect
   ```

3. Below the `[Output]` section, create a configuration section for the engine by typing the task name

inside square brackets. For example:

```
[XProtect]
```

4. In the new section, set the following parameters:

`Type`	The output engine to use. Set this parameter to **`milestoneoutput`**.
`Input`	A comma-separated list of the tracks that you want to output. To output information to Milestone, ensure you include the `Proxy` track generated by the Milestone ingest engine.
`ProxyTrack`	The `Proxy` track generated by the Milestone ingest engine.
`XSLTemplate`	The XSL template to use to transform the output track into a format that can be accepted by the Milestone system.
`Host`	The host name or IP address of the Milestone XProtect system.
`Port`	(Optional) The Milestone XProtect server port.
`GUID`	(Optional) The Milestone GUID of the camera that the events are associated with. This is not required if the video is ingested using the Milestone ingest engine.
`Location`	(Optional) Location metadata to send with the event.

For example:

```
[XProtect]
Type=milestoneoutput
Input=ANPR.Result,MilestoneIngest.Proxy
ProxyTrack=MilestoneIngest.Proxy
XSLTemplate=./xsl/toMilestone.xsl
Host=milestone-server
Port=9090
Location=Cambridge
```

For more information out the configuration parameters that you can use to configure this task, refer to the *Media Server Reference*.

5. Save and close the configuration file. If you modified the Media Server configuration file, you must restart Media Server for your changes to take effect.

Configure Milestone

To configure your Milestone surveillance system to process information sent by Media Server

1. Open the Milestone XProtect management application.
2. Make sure that the Milestone system has *Analytics Events* enabled, and is listening on the same port you specified in the Media Server configuration file.
3. For each type of event that you send to Milestone, add an *Analytic Event* to the Milestone system.

The name of the analytic event must match the message produced by Media Server.

- The default message for intelligent scene analysis events is the iSAS category name.
- The default message for all ANPR events is `ANPR`.
- The default message for recognized faces is the name of the face database.
- The default message for unrecognized faces is `NOT IN DATABASE`.

Tip: To modify the message produced by Media Server, modify the `toMilestone.xsl` XSL template. The `<message>` element in the XML sent to Milestone can contain any name, but the names of the analytic events that you create in Milestone must match the messages produced by Media Server.

Note: The names are case-sensitive. For example, if you have a category in your iSAS configuration called "JumpRedLight", create an Analytic Event of the same name.

4. Add *Alarm Definitions* to the Milestone system as necessary. Use the Analytic Events that you created as the *Triggering Events*.
5. In the *Alarm List Configuration (Advanced Configuration* > *Alarms* > *Alarm Data Settings)*, select all of the columns. This allows users of the Milestone XProtect Smart Client to view all of the metadata that is provided by Media Server.

For more information about how to configure your Milestone system, refer to the Milestone documentation.

Appendixes

This section contains the following Appendixes:

Appendix A: OCR Supported Languages

This appendix describes the languages that are supported by Media Server OCR.

Latin Alphabet

Afrikaans (af)	Esperanto (eo)	Irish (ga)	Romanian (ro)
Basque (eu)	Estonian (et)	Latin (la)	Slovak (sk)
Catalan (ca)	Finnish (fi)	Latvian (lv)	Slovenian (sl)
Croatian (hr)	French (fr)	Lithuanian (lt)	Spanish (es)
Czech (cs)	German (de)	Maltese (mt)	Swedish (sv)
Danish (da)	Hungarian (hu)	Norwegian (no)	Turkish (tr)
Dutch (nl)	Icelandic (is)	Polish (pl)	Welsh (cy)
English (en)	Italian (it)	Portuguese (pt)	

Cyrillic Alphabet

Bulgarian (bg)	Serbian (sr)
Macedonian (mk)	Ukrainian (uk)
Russian (ru)	

Other Alphabets

Arabic (ar), Persian (fa), Urdu (ur)

Simplified Chinese (zhs), Traditional Chinese (zht)

Greek (el)

Hebrew (he)

Japanese (ja)

Korean (ko)

Appendix B: OCR Supported Specialized Fonts

This appendix lists the specialized fonts supported by Media Server OCR.

Font and Character Set Codes

Font	Code
General	`auto`
OCR-A	`ocra`
OCR-B	`ocrb`
E13B	`e13b`
Farrington 7B	`fa7b`
Custom font used for Bloomberg Terminal GUI	`blmt`

Appendix C: ANPR Supported Locations

The Automatic Number Plate Recognition (ANPR) analysis engine supports reading number plates from the following locations.

Location	ISO-3166 code
Albania	AL
Algeria	DZ
Argentina	AR
Australia - New South Wales	AU-NSW
Australia - Queensland	AU-QLD
Australia - South Australia	AU-SA
Australia - Victoria	AU-VIC
Australia - Western Australia	AU-WA
Austria	AT
Bahrain	BH
Belgium	BE
Brazil	BR
Canada	CA
Colombia	CO
Czech Republic	CZ
Denmark	DK
Finland	FI
France	FR
Germany	DE
Greece	GR
India	IN
Indonesia	ID
Ireland	IE

Israel	IL
Italy	IT
Japan	JP
Kingdom of Saudi Arabia	SA
Kuwait	KW
Malaysia	MY
Mexico	MX
Netherlands	NL
Nigeria	NG
Norway	NO
New Zealand	NZ
Oman	OM
Peru	PE
Philippines	PH
Poland	PL
Portugal	PT
Qatar	QA
Russia	RU
Serbia	RS
Singapore	SG
Slovenia	SL
South Africa	ZA
Spain	ES
Sweden	SE
Switzerland	CH
Syria	SY
Thailand	TH
Turkey	TR

Ukraine	UA
United Arab Emirates - Abu Dhabi	AE-AZ
United Arab Emirates - Ajman	AE-AJ
United Arab Emirates - Dubai	AE-DU
United Arab Emirates - Fujairah	AE-FU
United Arab Emirates - Ras al-Khaimah	AE-RK
United Arab Emirates - Sharjah	AE-SH
United Arab Emirates - Umm al-Quwain	AE-UQ
United Kingdom	UK
United States - Alabama	US-AL
United States - Alaska	US-AK
United States - Arizona	US-AZ
United States - Arkansas	US-AR
United States - California	US-CA
United States - Colorado	US-CO
United States - Connecticut	US-CT
United States - Delaware	US-DE
United States - Florida	US-FL
United States - Georgia	US-GA
United States - Hawaii	US-HI
United States - Idaho	US-ID
United States - Illinois	US-IL
United States - Indiana	US-IN
United States - Iowa	US-IA
United States - Kansas	US-KS
United States - Kentucky	US-KY
United States - Louisiana	US-LA
United States - Maine	US-ME

United States - Maryland	US-MD
United States - Massachusetts	US-MA
United States - Michigan	US-MI
United States - Minnesota	US-MN
United States - Mississippi	US-MS
United States - Missouri	US-MO
United States - Montana	US-MT
United States - Nebraska	US-NE
United States - Nevada	US-NV
United States - New Hampshire	US-NH
United States - New Jersey	US-NJ
United States - New Mexico	US-NM
United States - New York	US-NY
United States - North Carolina	US-NC
United States - North Dakota	US-ND
United States - Ohio	US-OH
United States - Oklahoma	US-OK
United States - Oregon	US-OR
United States - Pennsylvania	US-PA
United States - Rhode Island	US-RI
United States - South Carolina	US-SC
United States - South Dakota	US-SD
United States - Tennessee	US-TN
United States - Texas	US-TX
United States - Utah	US-UT
United States - Vermont	US-VT
United States - Virginia	US-VA
United States - Washington	US-WA

United States - Washington, DC	US-DC
United States - West Virginia	US-WV
United States - Wisconsin	US-WI
United States - Wyoming	US-WY
Venezuela	VE

Appendix D: Pre-Trained Classifiers

HPE may provide classifiers that you can use with Media Server to classify images.

The following classifiers are currently available:

File name on download center	Description
`Imageserver-ObjectClass_ImageNet.dat`	A neural net (CNN) classifier that contains training for the Large Scale Visual Recognition Challenge (ILSVRC) classes listed at http://image-net.org/challenges/LSVRC/2012/browse-synsets.

For information about how to import a classifier into your training database, see "Import a Classifier" on page 146

Appendix E: Pre-Trained Object Detectors

HPE may provide pre-trained object detectors that you can use with Media Server to detect objects belonging to certain generic categories in images and videos.

The following detectors are currently available:

File name on download center	Description
`Mediaserver-ObjectDet_RoadScene.dat`	A neural net (CNN) object detector that contains training for detecting instances of cars, vans and people in images and videos. Single or multiple instances of each of these objects can be located in an image at a time.

For information about how to import a detector into your training database, see "Import a Detector" on page 150.

Glossary

A

ACI (Autonomy Content Infrastructure)
A technology layer that automates operations on unstructured information for cross-enterprise applications. ACI enables an automated and compatible business-to-business, peer-to-peer infrastructure. The ACI allows enterprise applications to understand and process content that exists in unstructured formats, such as email, Web pages, Microsoft Office documents, and IBM Notes.

ACI Server
A server component that runs on the Autonomy Content Infrastructure (ACI).

ACL (access control list)
An ACL is metadata associated with a document that defines which users and groups are permitted to access the document.

action
A request sent to an ACI server.

active directory
A domain controller for the Microsoft Windows operating system, which uses LDAP to authenticate users and computers on a network.

ANPR
Automatic Number Plate Recognition, which reads the number/license plate of a vehicle.

C

Category component
The IDOL Server component that manages categorization and clustering.

Community component
The IDOL Server component that manages users and communities.

connector
An IDOL component (for example File System Connector) that retrieves information from a local or remote repository (for example, a file system, database, or Web site).

Connector Framework Server (CFS)
Connector Framework Server processes the information that is retrieved by connectors. Connector Framework Server uses KeyView to extract document content and metadata from over 1,000 different file types. When the information has been processed, it is sent to an IDOL Server or Distributed Index Handler (DIH).

Content component
The IDOL Server component that manages the data index and performs most of the search and retrieval operations from the index.

D

DAH (Distributed Action Handler)
DAH distributes actions to multiple copies of IDOL Server or a component. It allows you to use failover, load balancing, or distributed content.

DIH (Distributed Index Handler)
DIH allows you to efficiently split and index extremely large quantities of data into multiple copies of IDOL Server or the

Content component. DIH allows you to create a scalable solution that delivers high performance and high availability. It provides a flexible way to batch, route, and categorize the indexing of internal and external content into IDOL Server.

I

IDOL
The Intelligent Data Operating Layer (IDOL) Server, which integrates unstructured, semi-structured and structured information from multiple repositories through an understanding of the content. It delivers a real-time environment in which operations across applications and content are automated.

IDOL Proxy component
An IDOL Server component that accepts incoming actions and distributes them to the appropriate subcomponent. IDOL Proxy also performs some maintenance operations to make sure that the subcomponents are running, and to start and stop them when necessary.

Intellectual Asset Protection System (IAS)
An integrated security solution to protect your data. At the front end, authentication checks that users are allowed to access the system that contains the result data. At the back end, entitlement checking and authentication combine to ensure that query results contain only documents that the user is allowed to see, from repositories that the user has permission to access.

K

KeyView
The IDOL component that extracts data, including text, metadata, and subfiles from over 1,000 different file types.

L

LDAP
Lightweight Directory Access Protocol. Applications can use LDAP to retrieve information from a server. LDAP is used for directory services (such as corporate email and telephone directories) and user authentication. See also: active directory, primary domain controller.

License Server
License Server enables you to license and run multiple IDOL solutions. You must have a License Server on a machine with a known, static IP address.

O

OmniGroupServer (OGS)
A server that manages access permissions for your users. It communicates with your repositories and IDOL Server to apply access permissions to documents.

P

primary domain controller
A server computer in a Microsoft Windows domain that controls various computer resources. See also: active directory, LDAP.

R

record
A single package of metadata in a track. A record produced by an analysis task might describe a recognized face, a word spoken in the audio, or a number plate detected by ANPR. A record can contain a significant amount of information; for example a record describing a number plate includes timestamps describing when the number

plate was detected, the position of the number plate in the video frame, the characters read from the number plate, the confidence score for recognition, and so on.

rolling buffer
A fixed-size storage area on disk where you can save encoded video on a continuous basis. When the rolling buffer is full, the oldest content is discarded to make space for the latest.

S

scene analysis
Scene analysis recognizes suspicious activity in video and produces alarms to alert security personnel. Scene analysis can be trained to recognize many suspicious events, including vehicles driving through red lights, people entering restricted areas, and abandoned bags and vehicles.

T

track
A stream of data produced by a processing task in Media Server. For example, when you ingest video the ingest task produces two tracks: one for video frames and the other for audio packets. Other tasks use these tracks. Analysis tasks read the data and produce tracks that contain analysis results; encoding tasks take the video and audio data to write files to disk. See also record.

V

View
An IDOL component that converts files in a repository to HTML formats for viewing in a Web browser.

W

Wildcard
A character that stands in for any character or group of characters in a query.

X

XML
Extensible Markup Language. XML is a language that defines the different attributes of document content in a format that can be read by humans and machines. In IDOL Server, you can index documents in XML format. IDOL Server also returns action responses in XML format.

Send Documentation Feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Media Server Administration Guide (Media Server 11.1)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to AutonomyTPFeedback@hpe.com.

We appreciate your feedback!