Media Server

Software Version 12.1

Administration Guide

MICRO FOCUS

Document Release Date: October 2018
Software Release Date: October 2018

Legal notices

Copyright notice

The only warranties for products and services of Micro Focus and its affiliates and licensors (“Micro Focus”) are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Documentation updates

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.

You can check for more recent versions of a document through the MySupport portal. Many areas of the portal, including the one for documentation, require you to sign in with a Software Passport. If you need a Passport, you can create one when prompted to sign in.

Additionally, if you subscribe to the appropriate product support service, you will receive new or updated editions of documentation. Contact your Micro Focus sales representative for details.

Support

Visit the MySupport portal to access contact information and details about the products, services, and support that Micro Focus offers.

This portal also provides customer self-solve capabilities. It gives you a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the MySupport portal to:

- Search for knowledge documents of interest
- Access product documentation
- View software vulnerability alerts
- Enter into discussions with other software customers
- Download software patches
- Manage software licenses, downloads, and support contracts
- Submit and track service requests
- Contact customer support
- View information about all services that Support offers

Many areas of the portal require you to sign in with a Software Passport. If you need a Passport, you can create one when prompted to sign in. To learn about the different access levels the portal uses, see the Access Levels descriptions.

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 or read the online Help.

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 install, configure, and start 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:

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

For more information about the types of analysis that you can run, see Analyze Media, on page 141.

Encode Media

Media Server can write a copy of ingested video to disk. You can encode video in several formats, segment the video, and change the size and bit rate of the video to create files that are optimized for your use case.

Media Server can also write video to 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.

Media Server can also write images to disk. You can encode copies of ingested images, or use the image encoder to encode still images from video. For example, if you run face recognition on a video file, you might want to encode images of the recognized faces.

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:

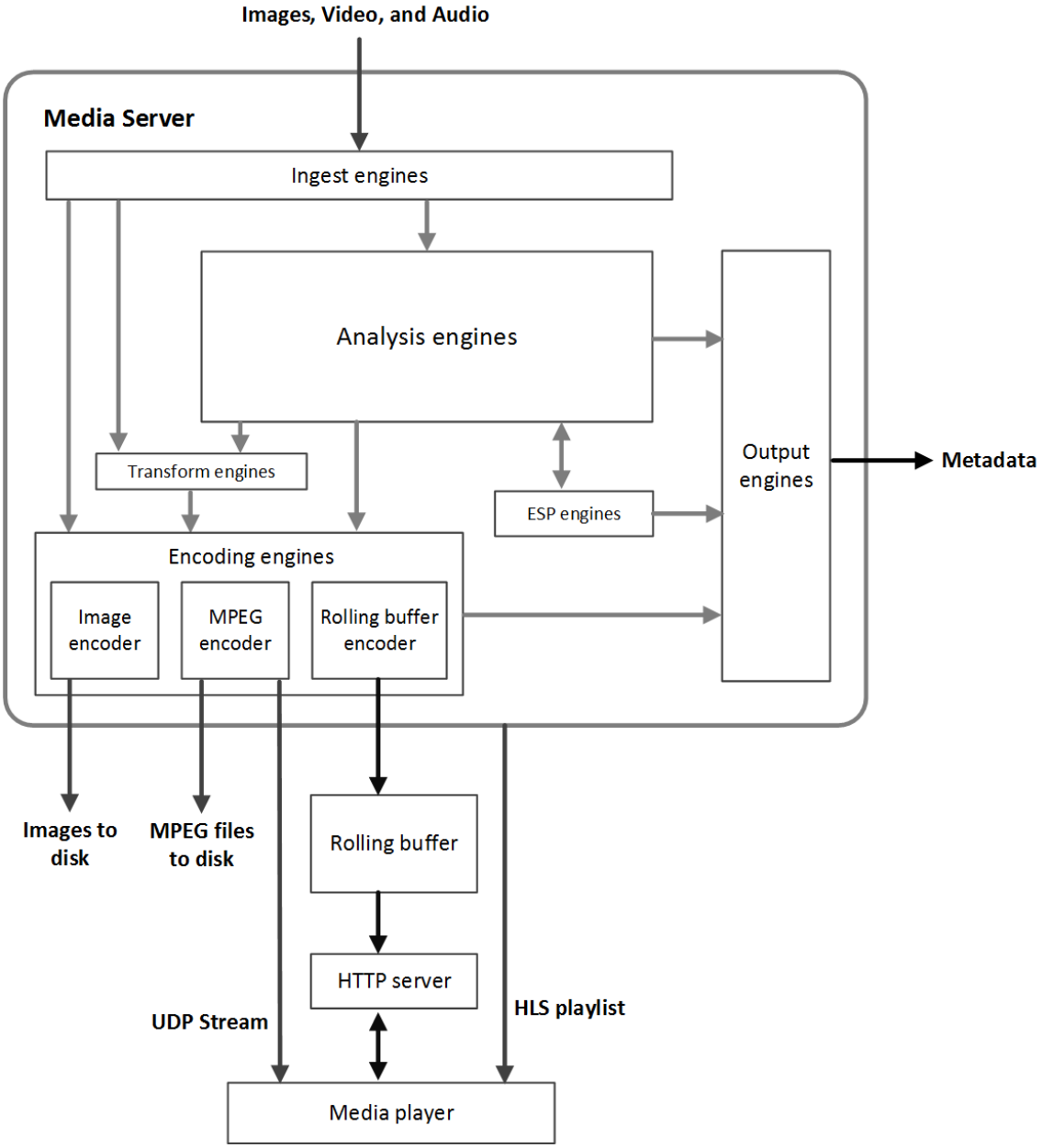
- Connector Framework Server (CFS)
- IDOL Server
- Vertica databases
- XML
- Milestone XProtect

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. 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.

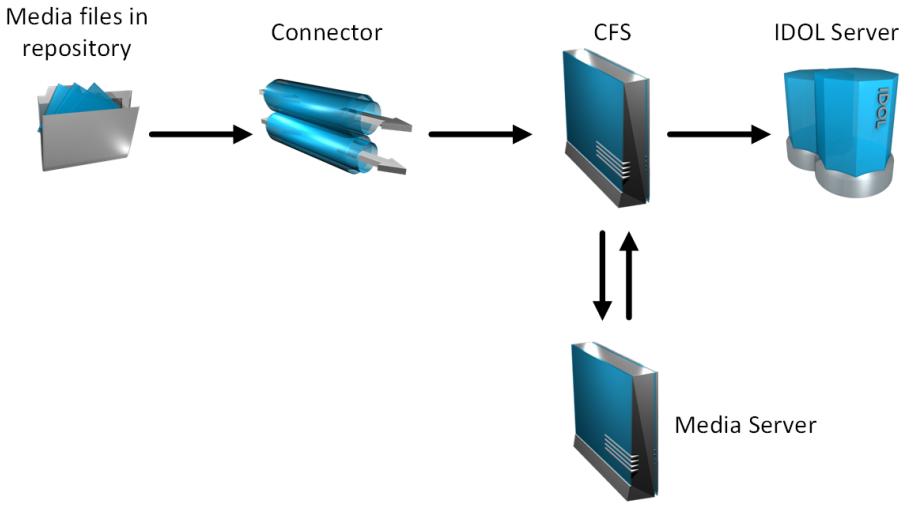
Information is passed between engines in the server. For example, an ingest engine decodes video and produces video frames and uncompressed audio for the other engines to use. Analysis engines run analysis on the decoded data. Output engines take information from the analysis engines and index it into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

The IDOL Platform

Media Server is one of the components in the *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 75.

- *Media Management and Analysis Platform Installation Guide*

  The 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 following table provides the minimum and recommended hardware specifications for running Media Server. The recommended specifications are only a guide, because actual requirements depend on the data that is being ingested and the processing tasks that you configure.

Component	Minimum	Recommended
CPU	The minimum number of CPU cores can be calculated as follows: Number of concurrent `process` actions x (1 core for ingest + 1 core for continuous encoding + 1 core for each analysis task) + 1 Continuous encoding is defined as encoding every frame, for example to a rolling buffer. To ingest video from four cameras simultaneously, run three analysis tasks on each stream, and encode video from each camera into a rolling buffer, the server should have a minimum of:	Micro Focus recommends Intel Xeon CPUs, because Media Server has been optimized for Intel processors.

	4 x (1 + 1 + 3) + 1 = 21 cores. To ingest video from four cameras simultaneously, run three analysis tasks on each stream, but encode only result images, the server should have a minimum of: 4 x (1 + 0 + 3) + 1 = 17 cores.	
Memory	4 GB	16 GB

IMPORTANT:
To run tasks that use convolutional neural networks on a machine that has a processor from the AMD Bulldozer series, download `libopenblas_AMD_Bulldozer.dll` from the MySupport portal, and rename it such that it replaces the file `libopenblas.dll` that is included in the Media Server installation. This is necessary to work around a known issue with these processors.

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

Antivirus Recommendations

If you are running antivirus software on the Media Server host machine, you must ensure for performance reasons that it does not monitor the Media Server directories.

Some advanced antivirus software can scan the network and might block some Media Server traffic, which can cause errors. Where possible, exempt the Media Server processes from this kind of network traffic analysis.

Memory Requirements

The amount of memory required by Media Server depends on many factors, including the type of tasks that you run. The following table provides minimum memory requirements for some analysis tasks.

The requirements listed below are the same regardless of the number of media sources that you process concurrently, and the number of threads that are used for each task.

Analysis Task	Memory Requirement
Face analysis	
Face detection	25 MB
Face recognition	65 MB
Face demographics	42 MB
Vehicle analysis	

Vehicle make recognition	43 MB
Image classification	
Image classification	55 MB
Additional memory required for the pre-trained ImageNet classifier	240 MB
Additional memory required for the pre-trained road scene classifier	50 MB
Object class recognition	
Object class recognition	18 MB
Additional memory required for the pre-trained person recognizer	340 MB
Additional memory required for the pre-trained road scene recognizer	338 MB
Optical Character Recognition	
OCR	28 MB
Additional memory required for Simplified Chinese text	40 MB
Additional memory required for Traditional Chinese text	53 MB
Additional memory required for Japanese text	31 MB
Additional memory required for Korean text	13 MB

Software Dependencies

The following software is required by Media Server:

- (Linux platforms only) Java Runtime Environment (JRE) 1.5 or later is required by KeyView to ingest some file types.

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

The minimum recommended versions of particular distributions are:

- CentOS 6
- Ubuntu 14.04

Supported Platforms with GPU support

- Windows x86 64. The only requirement for GPU Media Server is that NVIDIA driver 352.07 (or later) is installed on the machine, so any Windows operating system on which you can successfully install this driver is supported.
- Linux x86 64

The most fully tested versions of these platforms are:

Windows

- Windows Server 2012 R2

Linux

- Ubuntu 16.04
- Ubuntu 14.04

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:\MicroFocus\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
     a. On the License Server dialog box, click **Yes**, and then click **Next**. The Existing License Server dialog box opens.
     b. Specify the host and ACI port of your License Server, and then click **Next**.
   - To install a new instance of License Server
     a. On the License Server dialog box, click **No**, and then click **Next**. The Service Name dialog box opens.
     b. In the Service name box, type the name of the Windows service to use for the License Server, and then click **Next**. The License Server dialog box opens.
     c. Specify the ports that you want License Server to listen on, and then type the path to your IDOL license key file (`licensekey.dat`), which you obtained when you purchased Media Server, or click  and navigate to the location. 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.

   If you want to install a Windows service for Media Server, select the **Create Windows service** check box and choose a name for the service. If you do not install a Windows service you can create one later by following the procedure in Install an IDOL Component as a Service on Windows, on page 28.

   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 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 IDOL 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 Media Server from the ZIP Package

You can install Media Server from the standalone ZIP package. Use the ZIP package if you want to install Media Server without installing other IDOL components. If you need to install other IDOL components you can install Media Server using the IDOL Server installer (see Install Media Server on Windows, on page 24 or Install Media Server on UNIX, on page 26).

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 34.
4. (Optional) If you want to run Media Server as a Windows service, create the service by following the instructions in Install an IDOL Component as a Service on Windows, below.
5. Start Media Server by following the instructions in Start Media Server, on page 73.

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 34.
3. (Optional) If you want to run Media Server as a service, create the service by following the instructions in Install an IDOL Component as a Service on Linux, on page 30.
4. Start Media Server by following the instructions in Start Media Server, on page 73.

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, you might want to add the License Server as a dependency.

   For example:

```
Component.exe -install -servicename ServiceName -displayname "Component Display
Name" -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 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

> **NOTE:**
> If your setup has an externally mounted drive that Media Server uses, you might need to modify the init script. The installed init script contains examples for an NFS mount requirement.

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
     ```

   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).

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

2. Run the following commands to set the 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
     ```

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

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

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 the 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.

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 Micro Focus Big Data Download Center and install the latest patch, if one has been released:
   a. Download the latest patch from the Micro Focus Big Data Download Center. 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, on page 59.

Licenses

To use IDOL solutions, you must have a running License Server, and a valid license key file for the products that you want to use. Contact Micro Focus 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 35.

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 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 IDOL 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.

License-related error messages, continued

Error message	Explanation
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 Micro Focus 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 Micro Focus Big Data Support with the exact error message and your license file.
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 Micro Focus 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 Micro Focus 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 Micro Focus 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.

Enable GPU Acceleration

Media Server can use a graphics card (GPU) to perform some processing tasks. Using a GPU rather than the CPU can significantly increase the speed of training and analysis tasks that use Convolutional Neural Networks.

Tasks that benefit from a GPU are:

- Image classification.
- Face demographics analysis.
- Face recognition.
- Object class recognition.
- Vehicle make recognition.

GPU Requirements

To accelerate processing by using a GPU, your system must have a NVIDIA graphics card with CUDA compute capability version 3.0 to 6.1 (Kepler, Maxwell, and Pascal micro-architecture). All Quadro and Tesla series cards that meet this requirement are supported. GeForce GTX series cards that meet this requirement are supported, but only with headless Linux operating systems. Tegra series cards are not supported, but you can request support by contacting Micro Focus. Media Server has been tested with NVIDIA Quadro K6000, Quadro M6000, and Tesla K80 graphics cards.

The number and size of concurrent tasks that you can run on the GPU is constrained by the amount of memory available on the graphics card. To achieve the best performance, the amount of memory in the machine must match or exceed the amount of RAM available on the GPU(s). For example, if you have two GPUs and each has 12 GB RAM, the machine must have at least 24 GB RAM to use the full performance of the GPUs.

Only one Media Server can use each GPU. To run several instances of Media Server with GPU support on the same physical machine, the machine must have multiple GPUs. In the configuration file for each Media Server, you must set the `GPUDeviceID` parameter to specify which GPU to use (set this parameter to a different value for each Media Server).

If you are installing Media Server on a virtual machine, the virtual machine might need additional configuration to use the GPU successfully.

Install Media Server

Install Media Server as described in one of the following sections:

- Install Media Server on Windows, on page 24
- Install Media Server on UNIX, on page 26
- Install Media Server from the ZIP Package, on page 28

NOTE:
If you install GPU Media Server on Ubuntu, Micro Focus recommends that you install a headless version of Ubuntu server.

Install the NVIDIA CUDA Driver

To use GPU acceleration, you must install the NVIDIA CUDA driver:

- version 376.51 or later on Linux.
- version 375.26 or later on Windows.

This can be installed independently, or by installing the CUDA toolkit version 8.0. Micro Focus recommends installing the driver only because this is easier and faster, and only installs the required components.

To install the NVIDIA CUDA driver on Windows

- Download the driver from http://www.nvidia.co.uk/Download/index.aspx?lang=en-uk. If asked to choose a "Download Type", select "Optimal Driver for Enterprise" because this option provides stable, supported drivers. After downloading the driver, run the installation program as an administrator, and follow the on-screen instructions.

To install the NVIDIA CUDA driver on Linux

1. Verify that your machine is running a supported operating system. Run the following command:

   ```
   lsb_release -a
   ```

   The operating system is described:

   ```
   Distributor ID: Ubuntu
   Description:    Ubuntu 14.04.3 LTS
   Release:        14.04
   Codename:       trusty
   ```

2. Verify that a CUDA-compatible card is available. Run the following command:

   ```
   lspci | grep -i nvidia | grep -i VGA
   ```

   This should produce output similar to:

   ```
   0f:00.0 VGA compatible controller: NVIDIA Corporation GF106GL [Quadro 2000]
   (rev a1)
   28:00.0 VGA compatible controller: NVIDIA Corporation GK110GL [Quadro K6000]
   (rev a1)
   ```

   You must verify that one or more of these GPUs support CUDA Compute Capability version 3.0 to 6.1. In the previous example, the Quadro 2000 GPU is not supported.

3. Install the essential utilities required to install the NVIDIA driver, by running the following command.

```
sudo apt-get install build-essential
```

4. Install the NVIDIA driver by running the following commands.

```
sudo apt-get purge nvidia*
sudo add-apt-repository ppa:graphics-drivers/ppa
sudo apt-get update
sudo apt-get install nvidia-376
sudo reboot
```

5. Verify that the driver installation was successful by running the `nvidia-smi` command. The driver version must be reported as the following version (or later):

```
376.51
```

Configure the GPU (Windows only)

Perform this step only if you are running Media Server on Windows. This step is not necessary on Linux.

To use a GPU to accelerate Media Server processing tasks, you must place the GPU in TCC mode. In this mode the graphics card is used for computation only and does not provide output for a display. Unless you use TCC mode, the GPU does not provide adequate performance and can be slower than using a CPU. Many GPUs are not in TCC mode by default, so you must place the card in TCC mode using the `nvidia-smi` tool.

Configure Media Server

To configure Media Server to use a GPU

1. Open the Media Server configuration file in a text editor.
2. In the `[Server]` section, set the following configuration parameters:

   `CUDAVersion` Specifies whether to enable GPU support and which CUDA version to use. Set this parameter to `8`.

   `GPUDeviceID` (Optional) If the server has more than one GPU, set this parameter to the device ID of the GPU to use. You can find the device ID for each graphics card in the application log when Media Server starts. To find out which GPU device ID corresponds to which GPU, match the PCI Bus ID logged by Media Server to the PCI Bus ID output by the `nvidia-smi` command, which also provides the GPU name.

   For example:

```
[Server]
...
CUDAVersion=8
```

3. Save and close the configuration file.

Verify that Media Server is Using the GPU

To determine whether Media Server is using a GPU, start Media Server and open the application log. The log should show the following after Media Server starts:

```
GPU devices available for use
GPU deviceID 0 (Hex PCIBusId: ..)
GPU deviceID 1 (Hex PCIBusId: ..)
GPU deviceID 2 (Hex PCIBusId: ..)
GPU deviceID 3 (Hex PCIBusId: ..)
GPU mode used for MediaServer
GPU Memory: ... bytes free ... bytes total
GPU deviceID currently used: 2 (Hex PCIBusId: ..)
```

If installation was unsuccessful, Media Server does not start and logs the reason to the application log.

Install Speech-to-Text Language Packs

To run speech-to-text, you must install a language pack. There are more than 60 language packs available for Media Server. Language packs can contain hundreds of megabytes of data, so they are not included in the Media Server installation and must be downloaded separately.

> **TIP:**
> A language pack supports a single language and a single audio sample rate. For example, there is a language pack for processing US English (16kHz) and another for US English (8kHz). The 8kHz language packs are for processing telephony audio. For a list of available language packs, see Speech Analysis Supported Languages, on page 383.

To install a language pack

1. Download the language pack from the MySupport portal.
2. Extract the contents of the zip file into the folder *staticdata*`/speechtotext/`, where *staticdata* is the folder specified by the `StaticDataDirectory` parameter in the `[Paths]` section of the Media Server configuration file. The default value of this parameter is the `staticdata` folder in the Media Server installation directory.
3. To confirm that the language pack was installed successfully, start Media Server and run the action `ListSpeechLanguagePacks`. The response lists each language pack that is available, along with its supported sample rate.

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: Set up a Training Database

This section describes how to set up a database to store training data for analysis operations that require training.

Introduction

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.

Micro Focus 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 or later with PostgreSQL ODBC driver 09.03.0300 to 09.06.0200. • MySQL version 5.x with MySQL ODBC Connector 5.3 or later.
CentOS 6	• PostgreSQL version 9.1 or later with PostgreSQL ODBC driver 09.03.0300 to 09.06.0200 and unixODBC version 2.2.14 or later. • MySQL version 5.x with MySQL ODBC Connector 5.1 or later 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 86, 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. Micro Focus 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. If security is not a consideration you could grant all privileges, but the required privileges are:

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

   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;
   GRANT USAGE ON ALL SEQUENCES 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)**.

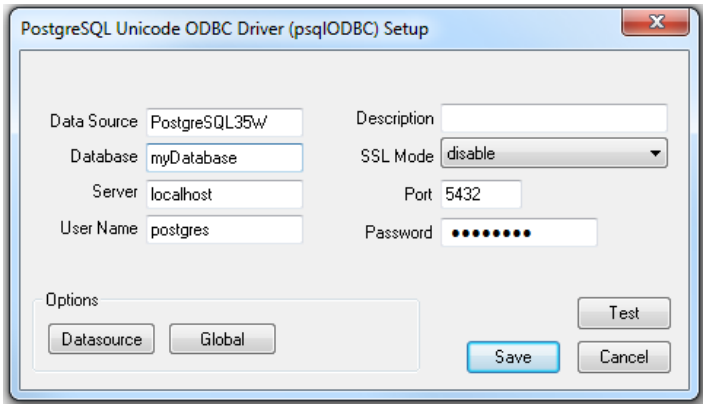
The ODBC Data Source Administrator dialog box opens.

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:

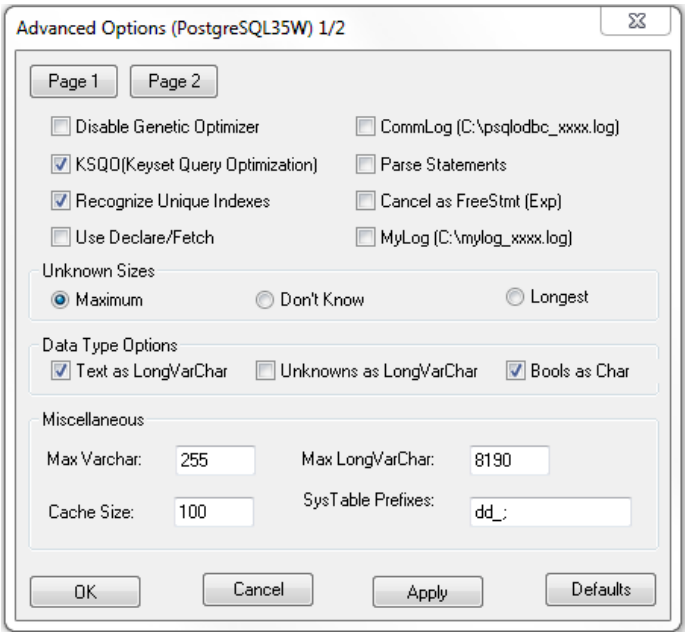
Data Source	The data source name (DSN). Media Server uses this string to connect to the database server.
Database	The name of the database that you created in Step 2.
Server	The IP address or hostname of the server that the database server is installed on.
User Name	The user name to connect to the database 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.

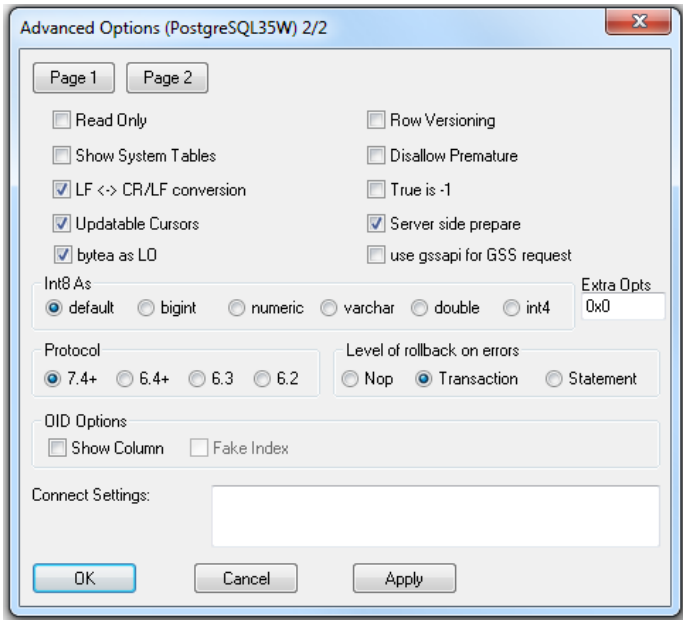
12. Click **Datasource**.

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



13. (Optional) Micro Focus 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 59).

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. Micro Focus 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. If security is not a consideration you could grant all privileges, but the required privileges are:

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

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;
GRANT USAGE ON ALL SEQUENCES 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) Micro Focus 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 59).

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**.

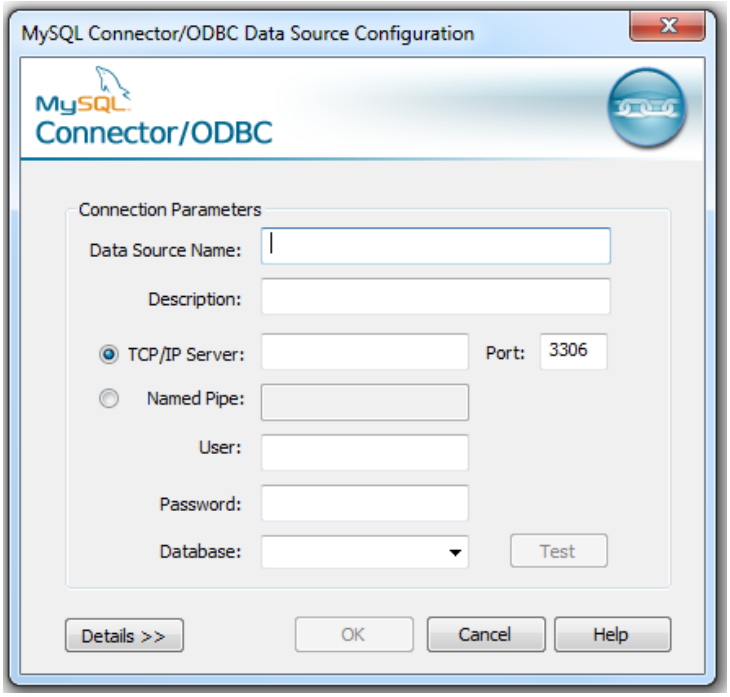
      The System and Security window opens.
   b. Click **Administrative Tools**.

      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 59).

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, on the next page).

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 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 a database that is hosted on an external database server, you must run an upgrade script when you upgrade Media Server. If you are using an internal database, any schema changes are applied automatically.

IMPORTANT:
If you are upgrading from Media Server 11.3 or earlier, and your training data is stored in the internal database, upgrade to Media Server 11.4, 11.5, or 11.6 and start Media Server before upgrading to Media Server 12.1.

Micro Focus 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
6	12.1	You are using the latest database schema
5	11.4	`my-upgrade_from_v5.sql` (for MYSQL databases) `postgres-upgrade_from_v5.sql` (for PostgreSQL databases)
4	11.2	`my-upgrade_from_v4.sql` (for MYSQL databases) `postgres-upgrade_from_v4.sql` (for PostgreSQL databases)
3	11.0	`my-upgrade_from_v3.sql` (for MYSQL databases) `postgres-upgrade_from_v3.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 3 to schema version 4, you can perform training with Media Server 11.0.x and Media Server 11.2.x. However, any training you perform with Media Server 11.2.x is available only to Media Server version 11.2.x, and any training that you perform using Media Server 11.0.x is available only to Media Server version 11.0.x.

After a successful schema upgrade you can remove data stored using earlier schema versions. This saves storage space on the database server. Micro Focus 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 to schema version 4, and do not want to use Media Server versions earlier than 11.2.x again, you can run `mysql-purge_before_v4.sql` or `postgres-purge_before_v4.sql` to remove schema version 3 and earlier 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 44
- Set Up a PostgreSQL Database on Linux, on page 48
- Set Up a MySQL Database on Windows, on page 52
- Set Up a MySQL Database on Linux, on page 55

> **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 12.1, 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_v`*`X`*`.sql`, or `postgres-purge_before_v`*`X`*`.sql`, where *X* is the oldest schema version you want to retain in the database.

Chapter 4: 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 (including the section headings) 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 of the external configuration file, in

quotation marks (""). You can use relative paths and network locations. After the configuration file path, 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 Parameters from an External Configuration File

This method allows you to import one or more parameters from an external configuration file at a specified point in your configuration file. You can import a single parameter or use wildcards to specify multiple parameters. The parameter values in the external file must match what you want to use in your file. This method does not import the section heading, such as `[License]` in the following examples.

To include parameters 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 parameters from the external configuration file.
3. On a new line, type a left angle bracket (<), followed by the path of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file path, add the name of the section 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
   ```

   You can use wildcards to import multiple parameters, but this method does not support default values. The * wildcard matches zero or more characters. The ? wildcard matches any single character. Use the pipe character | as a separator between wildcard strings. For example:

   ```
   < "license.cfg" [License] LicenseServer*
   ```

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

Micro Focus 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 -t`*`EncryptionType`* `[-oKeyFile] [-c`*`FILE`* `-s`*`SECTION`* `-p`*`PARAMETER`*`]` *`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.
`-o`*`KeyFile`*	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.

Configure Client Authorization

You can configure Media Server to authorize different operations for different connections.

Authorization roles define a set of operations for a set of users. You define the operations by using the `StandardRoles` configuration parameter, or by explicitly defining a list of allowed actions in the `Actions` and `ServiceActions` parameters. You define the authorized users by using a client IP address, SSL identities, and GSS principals, depending on your security and system configuration.

For more information about the available parameters, see the *Media Server Reference*.

> **IMPORTANT:**
> To ensure that Media Server allows only the options that you configure in `[AuthorizationRoles]`, make sure that you delete any deprecated *`Role`*`Clients` parameters from your configuration (where *`Role`* corresponds to a standard role name, for example `AdminClients`).

To configure authorization roles

1. Open your configuration file in a text editor.
2. Find the `[AuthorizationRoles]` section, or create one if it does not exist.
3. In the `[AuthorizationRoles]` section, list the user authorization roles that you want to create. For example:

   ```
   [AuthorizationRoles]
   0=AdminRole
   1=UserRole
   ```

4. Create a section for each authorization role that you listed. The section name must match the name that you set in the `[AuthorizationRoles]` list. For example:

   ```
   [AdminRole]
   ```

5. In the section for each role, define the operations that you want the role to be able to perform. You can set `StandardRoles` to a list of appropriate values, or specify an explicit list of allowed actions by using `Actions`, and `ServiceActions`. For example:

   ```
   [AdminRole]
   StandardRoles=Admin,ServiceControl,ServiceStatus

   [UserRole]
   Actions=GetVersion
   ServiceActions=GetStatus
   ```

   > **NOTE:**
   > The standard roles do not overlap. If you want a particular role to be able to perform all actions, you must include all the standard roles, or ensure that the clients, SSL identities, and so on, are assigned to all relevant roles.

6. In the section for each role, define the access permissions for the role, by setting `Clients`, `SSLIdentities`, and `GSSPrincipals`, as appropriate. If an incoming connection matches one of the allowed clients, principals, or SSL identities, the user has permission to perform the operations allowed by the role. For example:

   ```
   [AdminRole]
   StandardRoles=Admin,ServiceControl,ServiceStatus
   Clients=localhost
   SSLIdentities=admin.example.com
   ```

7. Save and close the configuration file.
8. Restart Media Server for your changes to take effect.

IMPORTANT:
If you do not provide any authorization roles for a standard role, Media Server uses the default client authorization for the role (localhost for `Admin` and `ServiceControl`, all clients for `Query` and `ServiceStatus`). If you define authorization only by actions, Micro Focus recommends that you configure an authorization role that disallows all users for all roles by default. For example:

```
[ForbidAllRoles]
StandardRoles=*
Clients=""
```

This configuration ensures that Media Server uses only your action-based authorizations.

Specify Modules to Enable

You can choose the modules to enable when Media Server starts. To use less memory and increase the speed at which Media Server starts, enable only the modules that you want to use.

Any module not listed below is always enabled and cannot be disabled.

To use any of the following modules you must specify the modules to enable when Media Server starts.

- **AudioCategorize**
- **AudioMatch**
- **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.
- **ImageClassification**
- **ImageComparison**
- **ImageHash**
- **NumberPlate** - number plate recognition.
- **ObjectClassRecognition**
- **ObjectRecognition**
- **OCR**
- **SpeakerID**
- **SpeechToText**
- **VehicleModel** - vehicle model recognition.

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. Use the `Enable` parameter to specify a comma-separated list of modules to enable. For example:

   ```
   [Modules]
   Enable=barcode,ocr
   ```

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.

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 5: 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:

  ```
  ServerName.exe -configfile configname.cfg
  ```

  where *ServerName* is the name of the server executable file, and 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 ServerName
    ```
  - On machines that use system V:

    ```
    service ServerName 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 ServerName`
  - On machines that use system V:

    `service ServerName 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 6: 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, below.

Micro Focus 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:**
> The `MaxInputString` and `MaxFileUploadSize` configuration parameters set a limit on the maximum size of HTTP strings and the maximum size of files that you can upload. The default values for these parameters allow HTTP strings that contain a maximum of 64,000 characters, and uploaded files with a maximum size of 10,000,000 bytes.

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, Micro Focus 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:14000 -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.

Monitor Asynchronous Actions using 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.

You can also use event handlers to monitor the size of asynchronous action queues. If a queue becomes full this might indicate a problem, or that applications are making requests to Media Server faster than they can be processed.

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

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

OnQueueEvent The `OnQueueEvent` handler is called when an asynchronous action queue becomes full, becomes empty, or the queue size passes certain thresholds.

- A `QueueFull` event occurs when the action queue becomes full.
- A `QueueFilling` event occurs when the queue size exceeds a configurable threshold (`QueueFillingThreshold`) and the last event was a `QueueEmpty` or `QueueEmptying` event.
- A `QueueEmptying` event occurs when the queue size falls below a configurable threshold (`QueueEmptyingThreshold`) and the last event was a `QueueFull` or `QueueFilling` event.
- A `QueueEmpty` event occurs when the action queue becomes empty.

Media Server supports the following types of event handler:

- The `TextFileHandler` writes event data to a text file.
- The `HttpHandler` sends event data to a URL.
- The `LuaHandler` runs a Lua script. The event data is passed into the script.

Configure an Event Handler

To configure an event handler, follow these steps.

To configure an event handler

1. Stop Media Server.
2. Open the Media Server configuration file in a text editor.
3. Set the `OnStart`, `OnFinish`, `OnError`, or `OnQueueEvent` parameter to specify the name of a section in the configuration file that contains the event handler settings.
   - 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 calls an event handler when the *`Example`* action starts and finishes successfully, and uses a different event handler to monitor the queue size:

     ```
     [Example]
     OnStart=NormalEvents
     OnFinish=NormalEvents
     OnQueueEvent=QueueSizeEvents
     ```

4. 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`, `OnError`, or `OnQueueEvent` parameter.
5. In the new section, set the `LibraryName` parameter.

   `LibraryName` The type of event handler 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.
   - To run a Lua script, set this parameter to **`LuaHandler`**, and then use the `LuaScript` parameter to specify the script to run. For information about writing the script, see Write a Lua Script to Handle Events, below.

   For example:

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

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

[QueueSizeEvents]
LibraryName=LuaHandler
LuaScript=./handle_queue_events.lua
```

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

Write a Lua Script to Handle Events

The Lua event handler runs a Lua script to handle events. The Lua script must contain a function named `handler` with the arguments `request` and `xml`, as shown below:

```
function handler(request, xml)
       ...
end
```

- `request` is a table holding the request parameters. For example, if the request was `action=Example&MyParam=Value`, the table will contain a key `MyParam` with the value `Value`. Some events, for example queue size events, are not related to a specific action and so the table might be empty.
- `xml` is a string of XML that contains information about the event.

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.

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:
Micro Focus 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 59.

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 Introduction, on page 42, 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=ActionStore`
	or
	`ConnectionString=Driver={PostgreSQL};`
	`Server=10.0.0.1; Port=9876;`
	`Database=SharedActions; Uid=user; Pwd=password;`

`MaxVarcharSize=0;`

If your connection string includes a password, Micro Focus recommends encrypting the value of the parameter before entering it into the configuration file. Encrypt the entire connection string. For information about how to encrypt parameter values, see Encrypt Passwords, on page 66.

> **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. Micro Focus 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.

Store Action Queues in Memory

Media Server provides asynchronous actions. Each asynchronous action has a queue to store requests until threads become available to process them. These queues are usually stored in a datastore file or in a database hosted on a database server, but in some cases you can increase performance by storing these queues in memory.

> **NOTE:**
> Storing action queues in memory improves performance only when the server receives large numbers of actions that complete quickly. Before storing queues in memory, you should also consider the following:
>
> - The queues (including queued actions and the results of finished actions) are lost if Media Server stops unexpectedly, for example due to a power failure or the component being forcibly stopped. This could result in some requests being lost, and if the queues are restored to a previous state some actions could run more than once.
> - Storing action queues in memory prevents multiple instances of a component being able to share the same queues.
> - Storing action queues in memory increases memory use, so please ensure that the server has sufficient memory to complete actions and store the action queues.

If you stop Media Server cleanly, Media Server writes the action queues from memory to disk so that it can resume processing when it is next started.

To configure Media Server to store asynchronous action queues in memory, follow these steps.

To store action queues in memory

1. Stop Media Server, if it is running.
2. Open the Media Server configuration file and find the `[Actions]` section.
3. If you have set any of the following parameters, remove them:
   - `AsyncStoreLibraryDirectory`
   - `AsyncStoreLibraryName`
   - `ConnectionString`
   - `UseStringentDatastore`
4. Set the following configuration parameters.

`UseInMemoryDatastore`	A Boolean value that specifies whether to keep the queues for asynchronous actions in memory. Set this parameter to **TRUE**.
`InMemoryDatastoreBackupIntervalMins`	(Optional) The time interval (in minutes) at which

the action queues are written to disk. Writing the queues to disk can reduce the number of queued actions that would be lost if Media Server stops unexpectedly, but configuring a frequent backup will increase the load on the datastore and might reduce performance.

For example:

```
[Actions]
UseInMemoryDatastore=TRUE
InMemoryDatastoreBackupIntervalMins=30
```

5. Save and close the configuration file.

   When you start Media Server, it stores action queues in memory.

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 Micro Focus Technical 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 7: 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 cannot configure ingestion, analysis, encoding, data transformation, event stream processing, or output in the Media Server configuration file (`mediaserver.cfg`). Instead, pass a task configuration to Media Server in the `process` action, when you start processing. This allows you to process media from different sources and run different analysis tasks for different types of media.

To pass a configuration to Media Server in 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.
- Create a configuration and save the file somewhere that is accessible to Media Server. When you run the `process` action to start processing, use the `ConfigPath` action parameter to specify the path 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.

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 (and might modify its schema). For example, you can change the size of video frames before sending them to the image encoder. 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.

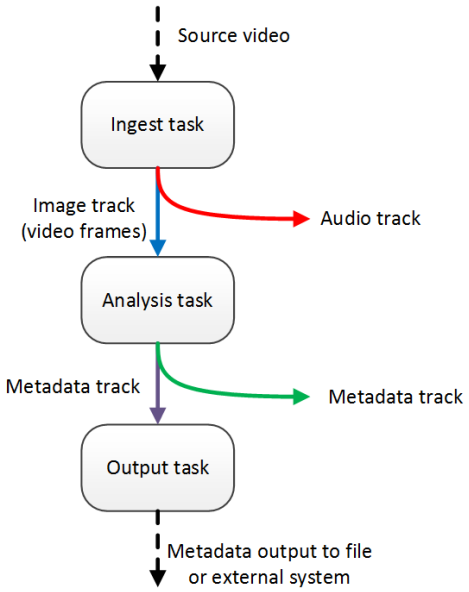
In a complete configuration, tasks that you define are connected. The output of an ingest task becomes the input for an analysis or encoding task. The output of an analysis task might become the input for an encoding, ESP, or output task. You can connect tasks in many different ways, which allows Media 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 many frames. For example, a person, object, or logo might appear on screen and remain there for several minutes. Media Server analyzes video frame by frame, but many analysis engines track events across frames because analyzing multiple frames can improve accuracy.

Analysis tasks can produce many different output tracks but, regardless of which track they belong to, records that relate to the same event always have the same ID.

- *Result* tracks contain records that summarize the analysis results for complete event. Each record can span many video frames and has a start time, peak time, end time, duration, and an ID. You can use the ID to find other records that are related to the same event. The purpose of a result track is to provide a summary of the analysis results that is suitable to output from Media Server. Media Server does not generate a record in a result track until an event has finished, because these records represent an entire event from beginning to end.

  Example: A face detection result track contains a single record for each detected face. Each record has a different ID.

  Example: A face recognition result track contains zero or more records for each detected face (there can be multiple recognition results when there are several matches that exceed the recognition threshold). Face recognition results inherit their ID from the detected face, so all of the recognition results for the same detected face have the same ID.

- *ResultWithSource* tracks are similar to result tracks because the records represent complete events. The records are the same as records in the result track, except that each record also includes the video frame that produced the best analysis result. For example, when you run face recognition the video frame with the highest confidence score is added to the record. This frame corresponds to the "peak" timestamp.

- *Data* tracks contain records that correspond to a single analyzed frame. A data track can contain hundreds of records that relate to the same event. A data track can also contain multiple records that relate to the same video frame, because multiple events can occur at the same time.

  Example: A face detection data track contains at least one record for every analyzed frame in which a face appears. If a person remains in the scene for several seconds, this track could contain hundreds of records that identify the same face and have the same ID. If a video frame contains three faces, the face detection data track will contain three records with timestamps matching that frame, each with a different ID.

- *DataWithSource* tracks are similar to data tracks because the records correspond to a single analyzed frame. The records are the same as the records in the data track, except that each record also includes the video frame that was analyzed.

> **TIP:**
> `Data` and `DataWithSource` tracks contain a lot of information, usually more than you want to output from Media Server. These tracks are intended to provide data for subsequent analysis tasks. For example, you can use the `DataWithSource` track from face detection as the input for face recognition, so that face recognition can analyze each face across multiple video frames.

- *Start* and *End* tracks contain records that describe the beginning or end of an event in the video.

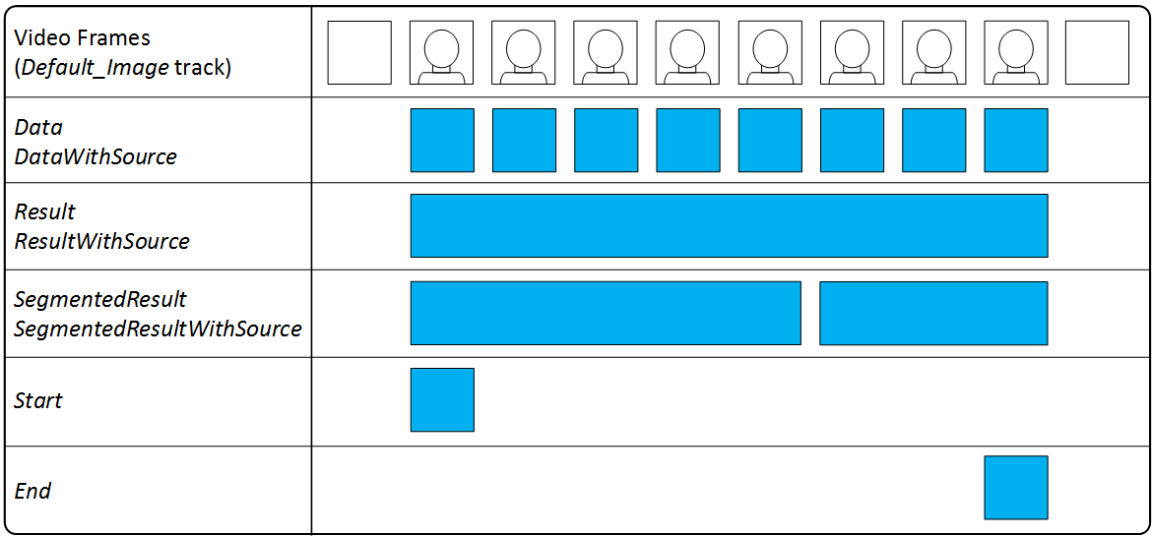
  Example: With face detection the start track contains a record when a face appears in the scene, and the end track contains a record when the face disappears.

  Example: Face recognition does not produce a start or end track, because information about events (detected faces) is provided by face detection.

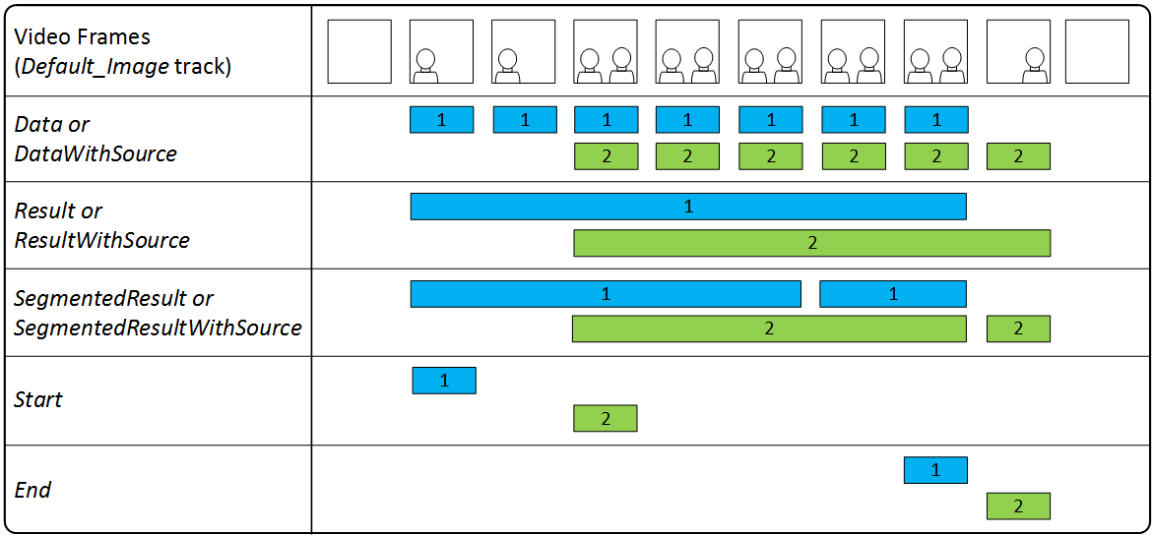
- *SegmentedResult* tracks are similar to result tracks, except that the maximum duration of a record is limited by a configuration parameter named `SegmentDuration`. When a record reaches the maximum duration, Media Server outputs the record and begins a new one with the same ID. This means that for every record in the result track that exceeds the maximum duration, there will be two or more records in the `SegmentedResult` track. Segmented results are useful when you need to obtain information about an event before it finishes.

- *SegmentedResultWithSource* tracks are similar to `SegmentedResult` tracks. The records are the same, except that each record also includes the best source frame that was available at the time the record was generated.

The following diagram shows how face detection creates records (represented by rectangles) when a face appears in a video.



The following diagram shows how face detection creates records (represented by rectangles) when two faces appear in a video. All of the records related to the same detected face (the same event) have the same ID. So, in the following example, all of the blue records (1) would have the same ID and all of the green records (2) would have the same ID.



In both of the previous examples:

- Media Server creates a single record in the `Result` and `ResultWithSource` tracks for each event (in this example a detected face). These records span the event and summarize the analysis results. When there are multiple people in the scene at the same time, the records overlap chronologically.
- The records in the `Data` and `DataWithSource` tracks correspond to a single analyzed frame. This means that there can be many records for each event. When there are multiple people in the scene, there are multiple records with timestamps matching the same video frame.
- Media Server creates a record in the `Start` track when a person appears in the scene.
- Media Server creates a record in the `End` track when a person leaves the scene.
- In these examples, each person remains in the scene longer than the configured `SegmentDuration`, so Media Server creates multiple records in the `SegmentedResult` and `SegmentedResultWithSource` tracks. Media Server starts a new record when the `SegmentDuration` is reached.

Some analysis tasks process the output of other engines. Face recognition, for example, processes records that are produced by face detection. You can see from the examples, above, that the face detection `DataWithSource` track provides much more information than the `ResultWithSource` track. When you configure face recognition, you can choose which track to process. Processing the `DataWithSource` track can result in better accuracy, because face recognition processes multiple video frames for each detected face. However, processing all of these frames is more computationally intensive and you should configure this only if your server has sufficient resources.

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

Create a Session 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 configuration must have a section named `[Session]`. This section contains a list of tasks that you want to run:

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=FaceRecognize
Engine3=EncodeImages
Engine4=OutputToIDOL
...
```

> **NOTE:**
> Task names cannot include any of the following characters:
>
> - period (`.`)
> - comma (`,`)
> - colon (`:`)
> - semicolon (`;`)
> - asterisk (`*`)
> - equals sign (`=`)

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

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

```
[Ingest]
Type=Video
```

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:

```
[Session]
Engine0=Ingest

[Ingest]
Type=Video
```

This example has a task named `Ingest`. The engine used to complete the task is specified by the `Type` parameter, in this case the `Video` 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 *taskName*`.Image_`*n*, where *taskName* is the name of the task and *n* is a unique number. Tracks are numbered from 1.
- Each audio track is named *taskName*`.Audio_`*lang*`_`*n*, where *taskName* is the name of the task, *lang* is the language, and *n* is a unique number. If the language is unknown, each track is named `Audio__`*n* (note the double underscore), where *n* is a unique number. The tracks are numbered from 1.

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

When you configure Media Server, the first image track produced by the ingest task can be specified by the aliases `Default_Image` and `Image_1`. The first audio track produced by an ingest engine can be specified by the alias `Default_Audio`.

Analysis

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 `OCR`. The task uses the `OCR` analysis engine:

```
[Session]
Engine0=Ingest
Engine1=OCR

[OCR]
Type=ocr
Input=Default_Image
```

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 `Default_Image` track produced by the ingest engine. The default value of the `Input` parameter is `Default_Image` for engines that require images and `Default_Audio` for engines that require audio, so in many cases you do not need to include the `Input` parameter in the configuration.

Some analysis engines require more complex input. For example, the face recognition analysis engine requires records that contain an image but also region information that specifies the position of the face in the image. The region information is not available in the `Default_Image` track from the ingest engine so you cannot use that track as the input and you must set the `Input` configuration parameter. The region information is provided by a face detection analysis task. The `DataWithSource` track produced by face detection includes the location of each face in every frame and the corresponding source images.

In the following example the input of the face recognition task is the `DataWithSource` track produced by the face detection task:

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=FaceRecognize

[FaceDetect]
Type=FaceDetect
...

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

The response to `action=ListEngines` describes the type of input required by each analysis engine. It also describes the output tracks that are produced. To be used as the input for a task, a track must provide at least the required record types. For more information about the output tracks produced by Media Server engines, refer to the *Media Server Reference*.

Transform

A configuration can contain any number of transformation tasks.

A transformation task requires a single input, transforms the data in some way (and might change its schema), and produces a single output. 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:

```
[Session]
Engine0=Ingest
Engine1=IdentifyKeyframes
Engine2=ScaleKeyframes

...

[ScaleKeyframes]
Type=Scale
```

```
Input=IdentifyKeyframes.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

A configuration can contain any number of encoding tasks. For example:

```
[Session]
Engine0=Ingest
Engine1=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

Usually a configuration contains at least one output task. The following example includes an output task named IDOL:

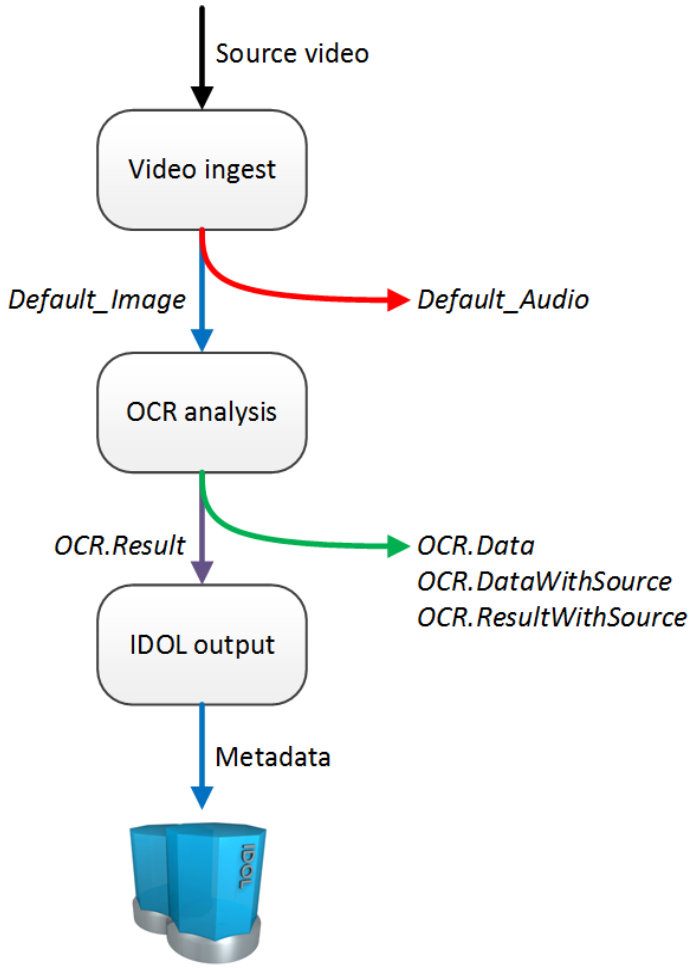
```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=FaceRecognize
Engine3=IDOL

[IDOL]
Type=IDOL
Input=FaceRecognize.Result
// idol engine options
```

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 Video ingest engine receives video from the source specified in the action. It 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.

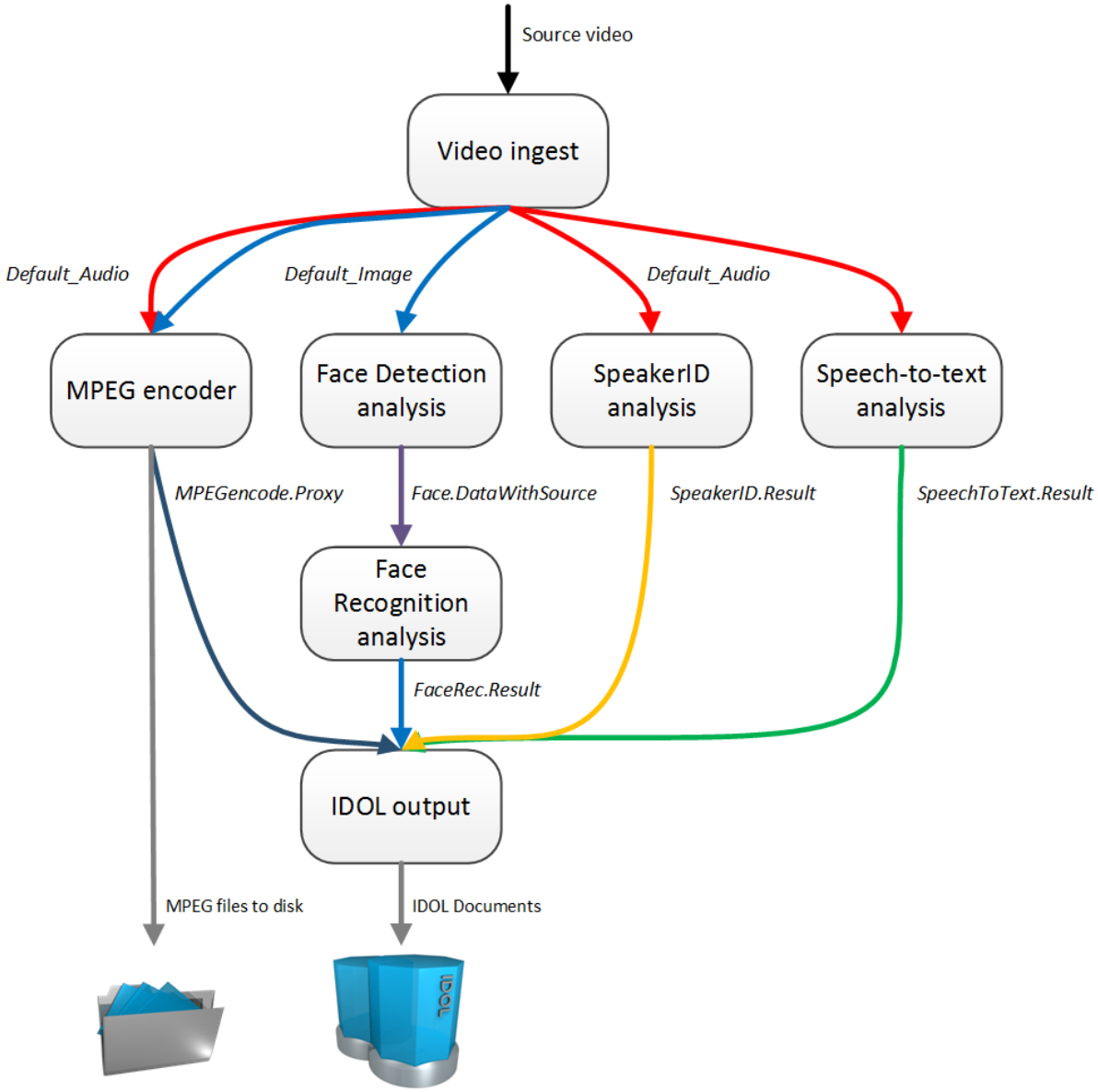
A Media Server configuration that matches this process is shown below.

- The `OCR` task includes the parameter `Input=Default_Image`, which specifies that the input for the OCR task is the first image track from the ingest engine. Setting this parameter is optional, because the default value of the `Input` parameter for engines that require images is `Default_Image`.
- The `IDOLOutput` task includes the parameter `Input=OCR.Result`. This specifies that the input for the IDOL output task is the `Result` track from the `OCR` task.

```
[Session]
Engine0=Ingest
Engine1=OCR
Engine2=IDOLOutput

[Ingest]
Type=Video

[OCR]
Type=OCR
Input=Default_Image

[IDOLOutput]
Input=OCR.Result
Type=IDOL
Mode=time
OutputInterval=60s
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 Video ingest engine receives video from the source specified in the action. It 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.

```
[Session]
Engine0=Ingest
Engine1=MPEGencode
Engine2=Face
Engine3=FaceRec
Engine4=SpeakerID
Engine5=SpeechToText
Engine6=IDOLOutput

[Ingest]
Type=Video

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

[Face]
Type=FaceDetect

[FaceRec]
Type=FaceRecognize
Input=Face.DataWithSource

[SpeakerID]
Type=SpeakerID
TemplateSet=speakers.ats

[SpeechToText]
Type=SpeechToText
Language=ENUK
SpeedBias=Live

[IDOLOutput]
```

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

Validate a Task Configuration File

You can use the action `ValidateProcessConfig` to check for errors in a task configuration file.

To validate a task configuration file

- Send the action `ValidateProcessConfig` to Media Server, including one of the following parameters:

`Config`	A base64 encoded configuration file to validate.
`ConfigName`	The name of a configuration file to validate, when the configuration file is stored in the directory specified by the `ConfigDirectory` parameter, in the `[Paths]` section of the Media Server configuration file.
`ConfigPath`	The path of a configuration file to validate.

  For example, to validate the configuration `facedetect.cfg`, which is stored in the directory specified by the `ConfigDirectory` parameter:

  `http://host:14000/action=ValidateProcessConfig&ConfigName=facedetect`

  The response to the action includes a `processable` element. This contains a Boolean value that indicates whether the configuration could be used to start processing. The value `true` indicates that there were no fatal errors. If you have used deprecated configuration parameters or the configuration contains unused parameters these are listed in the `errors` element of the response but are not considered to be fatal errors.

  A value of `true` does not guarantee that processing will succeed, because this might depend on the source media and whether resources (such as another Media Server) are available. If the configuration sends records to another Media Server, the action does not validate configurations on the downstream server.

  If errors are detected, they are described in the `errors` element of the response.

  For more information about this action, refer to the *Media Server Reference*.

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 demographics	✓	✓
Optical Character Recognition	✓	✓
Image classification	✓	✓
Object class recognition	✓	✓
Object recognition	✓	✓
Text detection	✓	✓
Number plate recognition	✓	✓
Vehicle make and model recognition	✓	✓
Clothing color analysis	✓	✓
Scene analysis	✗	✓
Keyframe extraction	✗	✓
Image comparison	✓	✓
Color clustering	✓	✓
Barcode recognition	✓	✓
Image hash generation	✓	✓
Audio analysis (audio categorization, language identification, speaker identification, speech-to-text, audio matching)	✗	✓
News segmentation	✗	✓
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	✗	✓
AndAny	✓	✓
AndThenAny	✗	✓
AndNot	✓	✓
AndNotThen	✗	✓
Or	✓	✓
Output	**Image**	**Video**
ACI Response	✓	✓
Files on disk	✓	✓
Connector Framework Server	✓	✓
IDOL Server	✓	✓
Vertica	✓	✓

ODBC database	✓	✓
HTTP POST	✓	✓
Milestone XProtect	✗	✓

Determine whether Media Server can Ingest Media

Media Server can analyze a media file or stream and return information about whether it can be ingested.

To determine whether Media Server can Ingest a File or Stream

- Run the action `DescribeMedia`. For example:

  ```
  /action=DescribeMedia&source=./media/my_video.ts
  ```

  Media Server returns information about the media and its ability to ingest it.

The `supported` element in the response indicates the likelihood that Media Server can ingest the media. The value can be:

- `yes` - the format and codecs are supported, so in most cases Media Server should be able to ingest the media. Ingestion can still fail in some cases, for example if Media Server encounters corrupt data.
- `maybe` - Media Server can open the media but the format or codecs have not been tested.
- `no` - indicates that the media cannot be ingested.

Related Topics

- Supported Audio and Video Codecs and Formats, on page 122
- Supported Image and Document File Formats, on page 129

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.

  From the following list of parameters, you must set either `Source` or `SourceData` to specify the source, and `Config`, `ConfigName`, or `ConfigPath` to specify the configuration to use.

  `Source` The media source to process. Specify one of the following:

  - a path to a file residing on a file system that Media Server can access.
  - the URL of a stream.
  - the name of a DirectShow video device (to ingest video from a DirectShow source, such as a capture card).
  - to ingest video from Milestone XProtect:
    - a camera name.
    - a camera UUID.
    - (Milestone XProtect Corporate only) a camera UUID followed by a stream ID, in the following format:

      `id:`*`camera_uuid`*`,streamid:`*`stream_id`*

      If your camera produces multiple streams and you do not specify a stream ID, Media Server ingests the stream that is configured as the default in the Milestone XProtect system.
  - to ingest video from Genetec Security Center, specify a camera logical ID.

  `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 77.

  `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` A base-64 encoded configuration that defines the tasks to run.

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

  `ConfigPath` The path of a configuration file that defines the tasks to run. Specify an absolute path or a path relative to the Media Server executable 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.

Monitor Progress

You can use the `QueueInfo` action to monitor the progress of Media Server as it processes a file (but not a stream).

For example, send the following action:

```
http://localhost:14000/action=QueueInfo
                                &QueueName=process
                                &QueueAction=progress
                                &Token=...
```

where the `token` parameter specifies the asynchronous action token returned by the `process` action.

Media Server returns a response similar to:

```
<autnresponse>
  <action>QUEUEINFO</action>
  <response>SUCCESS</response>
  <responsedata>
    <action>
      <token>.....................................</token>
      <status>Processing</status>
      <progress>
        <building_mode>false</building_mode>
        <percent>26.407</percent>
        <time_processing>71</time_processing>
        <estimated_time_remaining>198</estimated_time_remaining>
      </progress>
    </action>
  </responsedata>
</autnresponse>
```

The response includes the following information:

- The `building_mode` element specifies whether Media Server is building progress information. If the value here is `true`, Media Server is still analyzing the file to determine its length.
- The `percent` element specifies the progress of Media Server. In the previous example, Media Server has processed 26% of the file.
- The `time_processing` element indicates how long Media Server has spent processing the file so far.
- The `estimated_time_remaining` element provides an estimate of how long Media Server needs to complete processing the file.

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.

Synchronize with the Latest Training

Some analysis operations require training. When you train Media Server the training is added to the database, but is not used for analysis until Media Server has synchronized with the database. Synchronization loads the data into memory and makes it ready for use. Synchronization is necessary so that you can use any Media Server to perform training, even when several Media Servers share a single database.

Automatic Synchronization

The default configuration ensures that analysis uses the latest training data.

Before it starts a `process` action that requires training data, Media Server synchronizes with the database and waits for the synchronization to complete. This is because the `SyncDatabase` parameter - available for tasks that use training from the database - has a default value of `TRUE`.

Media Server also synchronizes with the database at regular intervals. You can configure the interval by setting the `SyncInterval` parameter in the `[Database]` section of the configuration file. This is important for processing video streams, because Media Server could continue processing the same stream for days or weeks. Automatic synchronization ensures that Media Server regularly loads the latest training without you having to intervene.

> **NOTE:**
> If you add a large amount of training data and immediately start a `process` action, Media Server might not use the latest training. This is because some training actions are asynchronous and the new training might not have been committed to the database before the `process` action starts. The new training therefore cannot be synchronized. To be sure that Media Server is using the latest training, wait for your training actions to complete before sending the `process` action.

> **NOTE:**
> With the Audio Match and Speaker ID analysis engines, synchronizing with the latest training has no effect on actions that are already running.

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.
- If you are processing a continuous video stream, you might not want Media Server to synchronize with the latest training while processing is in progress.
- 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.

The configuration parameter `ScheduledSync` specifies whether Media Server synchronizes with the database at regular intervals. The default value is `periodic`, which enables scheduled synchronization at intervals defined by the parameter `SyncInterval`. To load training data when Media Server starts, but not synchronize at regular intervals, set **`ScheduledSync=startup`**. To disable scheduled synchronization completely, set **`ScheduledSync=never`**. In this case, Media Server does not load any training data when it starts and does not synchronize with the database at timed intervals.

To prevent Media Server synchronizing with the database before it starts a `process` action, set **`SyncDatabase=FALSE`** in the configuration section for each relevant analysis task. With this setting, be aware that `process` requests could use outdated training.

If you disable automatic synchronization but want to load the latest training, you can run an action (see Actions to Load and Unload Training, below).

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, classifiers, and so on, this can consume a significant amount of memory.

To load only the training you need, set **`ScheduledSync=startup`** or **`ScheduledSync=never`** to disable scheduled synchronization. If you set `ScheduledSync=startup`, Media Server loads all training data when it starts but you can remove training from memory as required. If you set `ScheduledSync=never`, Media Server does not load any training data when it starts, so you can load only the training data you need.

To load training or remove training from memory, use one of the following methods:

- Allow Media Server to synchronize with the latest training before beginning an analysis task (this occurs by default). Media Server loads the training it needs to complete the task.
- To prevent Media Server synchronizing with the training database before beginning a task, set **`SyncDatabase=FALSE`** in the `[`*`TaskName`*`]` section for your analysis task. In this case, you, must manually synchronize the training data you need by running actions (see Actions to Load and Unload Training, below).

Actions to Load and Unload Training

The following table lists the actions that you can use when you have disabled automatic synchronization but want to synchronize with the latest training, or remove training from memory.

Analysis task	Action to load training	Action to unload training
Audio matching	`SyncAudioMatchClips`	
Face recognition	`SyncFaces`	`UnsyncFaces`
Image classification	`SyncClassifiers`	`UnsyncClassifiers`
Image hashing	`SyncImageHashes`	`UnsyncImageHashes`
Object class recognition	`SyncObjectClasses`	`UnsyncObjectClasses`
Object recognition	`SyncObjects`	`UnsyncObjects`
Speaker identification	`SyncSpeakers`	
Speech-to-text (custom language models)	`SyncCustomSpeechLanguageModels`	
Vehicle model ID	`SyncVehicleModels`	`UnsyncVehicleModels`

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 skip frames. Allowing Media Server to use more than one thread means that it can process more frames and this can increase accuracy.
- If you have configured Media Server to process a video file without skipping 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.
- face recognition.
- face demographics analysis.
- face expression analysis.
- clothing analysis.
- optical character recognition.
- object class recognition.
- object recognition.
- number plate recognition.
- image classification.

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
```

> **NOTE:**
> If you have installed Media Server with GPU support and are configuring a task that can use the GPU, set `NumParallel=1` and use the configuration parameter `GPUNumParallel` to specify the number of video frames to process concurrently on the GPU. For more information, see

> Optimize Analysis Performance with a GPU, below.

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.

Optimize Analysis Performance with a GPU

Media Server can use a graphics card (GPU) to perform some processing tasks. Using a GPU rather than the CPU can significantly increase the speed of analysis tasks that use Convolutional Neural Networks.

This section describes how to configure the following tasks to achieve optimum performance when you accelerate processing using a GPU.

- Face recognition.
- Face demographics analysis.
- Image classification.

When you are using a GPU to accelerate processing, configure your analysis task with `NumParallel=1`. To specify the number of video frames to process concurrently on the GPU, set the parameter `GPUNumParallel`. The value of this parameter must be a power of 2, such as 4, 8, 16, 32, 64, and so on.

```
[Demographics]
Type=Demographics
Input=FaceDetect.DataWithSource
NumParallel=1
GPUNumParallel=32
```

You should choose the highest possible value for `GPUNumParallel`; the limit is the amount of memory available on your GPU. If you set a value that is too high and the GPU runs out of memory, analysis will fail. You can monitor the amount of GPU memory used by Media Server with the `nvidia-smi` tool.

If you are processing low numbers of frames, Media Server might send video frames to the GPU before there are enough for a complete batch. You can configure the amount of time that Media Server waits for video frames by setting the configuration parameter `GPUBatchingDuration`. To maximize throughput and use the GPU most efficiently, configure a long `GPUBatchingDuration` so that Media Server waits until there is a full batch of video frames to analyze. If you are processing low volumes of frames or require the analysis results as rapidly as possible, you can reduce the duration.

If your analysis task supports the configuration parameter `SampleInterval`, you can decrease the interval. (For more information about sample intervals, see Choose the Rate of Ingestion, on page 123). Analysis with a GPU is significantly faster than with a CPU and so Media Server can process significantly more frames. In the case of face recognition and demographics analysis, which process the output of another analysis task, set the `SampleInterval` parameter on the first analysis task (face detection).

```
[FaceDetect]
Type=FaceDetect
// NumParallel is set because Face Detection uses the CPU for analysis
NumParallel=8
FaceDirection=Front
MinSize=200
SizeUnit=pixel
SampleInterval=0ms

[Demographics]
Type=Demographics
Input=FaceDetect.DataWithSource
NumParallel=1
GPUNumParallel=32
```

In this example, face detection attempts to process every frame because `SampleInterval=0ms`. Every frame that contains a detected face is written to the `FaceDetect.DataWithSource` track, and the demographics analysis task processes these frames using the GPU.

Optimize Performance when Processing Images

When Media Server begins an analysis task it might have to load information. For example, if you start processing and your configuration includes OCR, Media Server must load language data for the languages that you have chosen. You can optimize performance by configuring Media Server to load the data when it starts and keep the data in memory. Throughput is improved because the data required is already stored in memory, and the overhead of loading data at the start of each task and unloading it at the end is removed. This can significantly increase throughput when you run many `process` actions that complete quickly, for example when you are processing batches of images or individual frames extracted from a video. Loading data when Media Server starts does not significantly improve throughput when you are processing video files or streams.

> **TIP:**
> Media Server automatically loads training data for analysis tasks such as face recognition or object recognition. For information about customizing how training data is loaded, see the relevant sections in Analyze Media, on page 141.

Number Plate Recognition

To load number plate formats when Media Server starts, and keep the data in memory, set the `NumberPlateLocation` parameter in the `[PersistentData]` section of the configuration file:

```
[PersistentData]
NumberPlateLocation=US-ME
```

This parameter accepts a comma-separated list of ISO-3166 location codes. To obtain a list of supported locations with ISO-3166 codes, use the action `ListNumberPlateLocations`.

> **NOTE:**
> If you set this parameter then all number plate recognition tasks must be configured with the same location (when you configure the number plate analysis task, set the `Location` parameter to the same value).

OCR

To load OCR languages when Media Server starts, and keep the data in memory, set the `OCRLanguages` parameter in the `[PersistentData]` section of the configuration file:

```
[PersistentData]
OCRLanguages=en,fr
```

This parameter accepts a comma-separated list of ISO 639-1 language codes. For a list of supported languages and their language codes, see OCR Supported Languages, on page 376.

You must still set the `Languages` parameter when you configure an OCR task. If an OCR task requires languages that are not pre-loaded, the languages are loaded when the task begins and unloaded when the task ends.

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 configure Media Server to ingest media.

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

Chapter 8: 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 Audio and Video 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 Video ingest task (see Ingest Video from a File, on page 125).

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. Micro Focus 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 frames at the sample interval that you have requested. In this case, an analysis task will automatically increase its sample interval to ensure that analysis keeps up with the video. This might cause a reduction in accuracy because the frames that are analyzed will be further apart, making it more difficult for Media Server to track objects across frames and reducing the effectiveness of *integration*. When you are processing a live stream, you cannot change the rate at which video is ingested, so to prevent the sample interval being increased the only solution is to increase the resources (particularly 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 the requested sample interval, it will automatically increase its sample

interval and analyze fewer frames.

- `0` - The rate at which video is ingested is determined by the slowest analysis task that you have configured. The interval between the frames selected for analysis will always be as close as possible to `SampleInterval`. 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 the actual sample interval being closer to the requested interval, or 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	Micro Focus 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 intervals should provide good accuracy in most cases. If necessary, Media Server will increase the sample interval to ensure that analysis occurs at normal (playback) speed. If Media Server does not have sufficient resources, it will increase the sample interval, which might cause a reduction in accuracy.	Media Server attempts to analyze every frame, but frames are skipped if the analysis tasks you have configured cannot process the frames at normal (playback) speed. Micro Focus does not recommend these settings because processing every frame is usually not necessary and Media Server is likely to skip large numbers of frames.
IngestRate=0	Micro Focus 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	Ensures that all frames are analyzed. Micro Focus does not recommend these settings because in most cases this will take much longer and might not increase accuracy. Use this mode only if it is critical that every frame is analyzed.

	sample intervals should provide good accuracy in most cases. If analysis is slow, Media Server ingests the file at a slower rate so that frames can be analyzed at the requested sample interval. If Media Server does not have sufficient resources, it will take longer to process the file, but accuracy is maintained.	

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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, set the following configuration parameters:

   `IngestRate` The rate at which Media Server ingests video. If you are ingesting video from files, Micro Focus recommends setting this parameter to **`0`**. For more information about choosing an ingest rate, see Choose the Rate of Ingestion, on page 123.

   `Engine`*`N`* The name of a section in the configuration file that will contain the ingestion settings

   For example:

   ```
   [Session]
   IngestRate=0
   Engine0=Ingest
   ```

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

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

   `IngestDateTime` (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 milliseconds or ISO 8601 UTC.

   `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:

```
[Ingest]
Type=Video
```

> **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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` 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`.
`Engine`*`N`*	The name of a section in the configuration file that will contain the ingestion settings

For example:

```
[Session]
IngestRate=1
Engine0=Ingest
```

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 **`Video`**.
`IngestDateTime`	(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 milliseconds or ISO 8601 UTC.
`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:

```
[Ingest]
Type=Video
```

> **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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 9: 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 107.

Supported Image and Document File Formats

Media Server can ingest all standard image formats:

- TIFF
- JPEG
- JPEG 2000
- PNG
- GIF (Media Server only ingests the first frame of an animated GIF)
- BMP (compressed BMP files are not supported) and ICO
- PBM, PGM, and PPM
- WebP

Additionally, Media Server uses IDOL 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 ingest image files or documents

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ```
3. Create a new section to contain the task settings, and set the following parameters:

`Type`	The ingest engine to use. Set this parameter to **`image`**.
`StartPage`	(Optional) The page in the image or document at which to start ingesting.
`MaximumPages`	(Optional) The maximum number of pages to ingest from a multi-page image or document.

For example:

```
[Ingest]
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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Output Records

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

Image Data

The image ingest engine writes image data to a track named `Image_1`.

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>...</imagedata>
      <width>222</width>
      <height>140</height>
      <pixelAspectRatio>1:1</pixelAspectRatio>
      <format>PNG</format>
      <compressionQuality>100</compressionQuality>
    </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 information about the image.
  - The `imagedata` element contains the image data, base-64 encoded.
  - The `width` and `height` elements provide the size of the image.
  - The `pixelAspectRatio` element describes the shape of the pixels that make up the image, for example 1:1 pixels are square.
  - The `format` element describes the format of the image data contained in the `imagedata` element. For images in the `Image_1` track, this value is always `PNG`.
  - The `compressionQuality` element describes the amount of compression that is applied to the data in the `imagedata` element. For images in the `Image_1` track, this value is always `100` (indicating maximum quality and no compression).

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>...</imagedata>
      <width>892</width>
      <height>1260</height>
      <pixelAspectRatio>1:1</pixelAspectRatio>
      <format>PNG</format>
      <compressionQuality>100</compressionQuality>
    </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.

Source Information

The image ingest engine produces a proxy track, named *taskName*`.proxy`, where *taskName* is the name of your ingest task. The purpose of the `proxy` track is to contain information about the ingested source. The engine produces one record in this track for each page in the ingested image or document.

The following XML shows a sample record:

```
<record>
  <pageNumber>1</pageNumber>
  <trackname>ImageIngestTask.Proxy</trackname>
  <proxy path="./image.jpeg" url="./image.jpeg" mimeType="image/jpeg"
   estimatedDuration="0" pages="1">
    <streams>
      <videoStream id="0" width="2592" height="1936" sar="1:1" codec=""/>
    </streams>
    <metadata>
      <tag name="Author">A Name</tag>
      <tag name="Creation Date">2014-04-09T09:15:19Z</tag>
      <tag name="Flash">Flash did not fire</tag>
      <tag name="GPS Latitude">52° 13' 10.69"</tag>
      <tag name="GPS Longitude">0° 8' 49.23"</tag>
      ...
    </metadata>
  </proxy>
</record>
```

The `metadata` element contains any metadata that Media Server was able to extract from the source. The information present in this element varies based on the format of the source file and the information present in the source.

Chapter 10: Cameras and Third-Party Systems

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

Ingest MJPEG Video streamed over HTTP

This section describes how to ingest MJPEG video that is streamed over HTTP.

To ingest MJPEG video that is streamed over HTTP

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ```

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

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

   For example:

   ```
   [Ingest]
   Type=MJPEG_HTTP
   ```

   > **TIP:**
   > There is no configuration parameter to specify the source. You must specify the URL 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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Ingest MxPEG Video from a File or Stream

Media Server can ingest MxPEG video (but not the audio) from a file or stream.

To ingest MxPEG video

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, set the following configuration parameters:

   `Engine`*`N`* The name of a section in the configuration file that will contain the ingestion settings.

   `IngestRate` To ingest MxPEG video from a stream, set this parameter to **`1`**.

   To ingest MxPEG video from a file set this parameter to **`0`**. `IngestRate=1` is not supported with MxPEG files.

   For example:

   ```
   [Session]
   Engine0=Ingest
   IngestRate=1
   ```

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 **`MxPEG`**.

   For example:

   ```
   [Ingest]
   Type=MxPEG
   ```

   > **TIP:**
   > There is no configuration parameter to specify the source. You must specify the path of the file or the URL 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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ```

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

`Type`	The ingest engine to use. Set this parameter to **`Video`**.
`Format`	The video format. Set this parameter to **`dshow`**.
`IngestDateTime`	(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 milliseconds or ISO 8601 UTC.

   For example:

   ```
   [Ingest]
   Type=Video
   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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Obtain a List of Device Names

Media Server can ingest video directly from a DirectShow device, such as a video capture card or camera. This section describes how to obtain the name of your device, so that you use the name in the `source` action parameter, when you start the `process` action.

> **NOTE:**
> Ingestion from a DirectShow device is supported only on Windows.

To obtain a list of DirectShow devices

1. Download and install FFmpeg.
2. In a command-line window, run the following command:

   ```
   ffmpeg -list_devices true -f dshow -i dummy
   ```

   FFmpeg lists the devices that are available, for example:

   ```
   C:\MediaServer_11.3.0>ffmpeg -list_devices true -f dshow -i dummy
   ffmpeg version ... Copyright (c) 2000-2016 the FFmpeg developers
     ...
   [dshow @ 000000000050a400] DirectShow video devices (some may be both video and
   audio devices)
   [dshow @ 000000000050a400]  "HP HD Webcam [Fixed]"
   [dshow @ 000000000050a400]     Alternative name ...
   [dshow @ 000000000050a400] DirectShow audio devices
   [dshow @ 000000000050a400]  "Internal Microphone Array (IDT "
   [dshow @ 000000000050a400]     Alternative name ...
   [dshow @ 000000000050a400]  "Stereo Mix (IDT High Definition"
   [dshow @ 000000000050a400]     Alternative name ...
   ```

   In this case you could ingest video from a device named `HP HD Webcam [Fixed]`. The name must be URL-encoded in the `process` action, for example:

   ```
   http://mediaserver:14000/a=process&configname=myconfig
                                    &source=HP%20HD%20Webcam%20%5BFixed%5D
   ```

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. Micro Focus recommends that you encrypt passwords before entering them into the configuration file. For information on how to do this, see Encrypt Passwords, on page 66.

To ingest video from Milestone XProtect

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ```

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:

```
[Ingest]
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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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. For more information about how to start processing, see Start Processing, on page 110.

Ingest Video from Genetec Security Center

Media Server can ingest live video from Genetec Security Center.

> **NOTE:**
> This feature is available only when you run Media Server on Windows.

Before you begin

1. Install the Genetec Security Center SDK, version 5.6, on the same machine as Media Server.
2. In the `[Paths]` section of the Media Server configuration file, set the configuration parameter `GenetecSDKDirectory` to the directory that contains the SDK, for example:

```
[Paths]
...
GenetecSDKDirectory=C:\Genetec_Security_Center_5.6_SDK
```

3. Restart Media Server.

To ingest video from Genetec

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

```
[Session]
Engine0=Ingest
```

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 **`GenetecIngest`**.
`RecorderHost`	The host name or IP address of the Genetec Security Center recording server.
`RecorderPort`	The port of the Genetec Security Center recording server.
`BasicUsername`	The user name to use to retrieve video from Genetec Security Center.
`BasicPassword`	The password to use to retrieve video from Genetec Security Center. Micro Focus recommends encrypting user names and passwords before entering them in configuration files. For information about how to do this, see Encrypt Passwords, on page 66.

For example:

```
[Ingest]
Type=GenetecIngest
RecorderHost=Genetec
RecorderPort=5500
BasicUsername=user
BasicPassword=encrypted_password
```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

You can now start ingesting video. When you use the `process` action, you must set the `source` parameter to the logical ID of a camera. For information about how to find the logical ID, refer to the Genetec Security Center documentation. For more information about how to start processing, see Start Processing, on page 110.

Ingest Video from VMS

Media Server can ingest video from a Micro Focus Video Management Server.

To ingest video from VMS

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ```

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

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

For example:

```
[Ingest]
Type=VMS
```

> **TIP:**
> There is no configuration parameter to specify the source. You must specify the URL of the stream 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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Part III: Analyze Media

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

- Face Detection, Recognition, and Demographics
- Optical Character Recognition
- Image Classification
- Object Class Recognition
- Object Recognition
- Text Detection
- Number Plate Recognition
- Vehicle Make and Model Recognition
- Clothing Color Analysis
- Scene Analysis
- Extract Keyframes
- Image Comparison
- Color Clustering
- Barcode Recognition
- Generate Image Hashes
- Audio Categorization
- Language Identification
- Speaker Identification
- Speech-to-Text
- Audio Matching
- Segment Video into News Stories

Chapter 11: Face Detection, Recognition, and Demographics

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.

> **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 145.

Detect Faces

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

To detect faces in media

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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
   ```

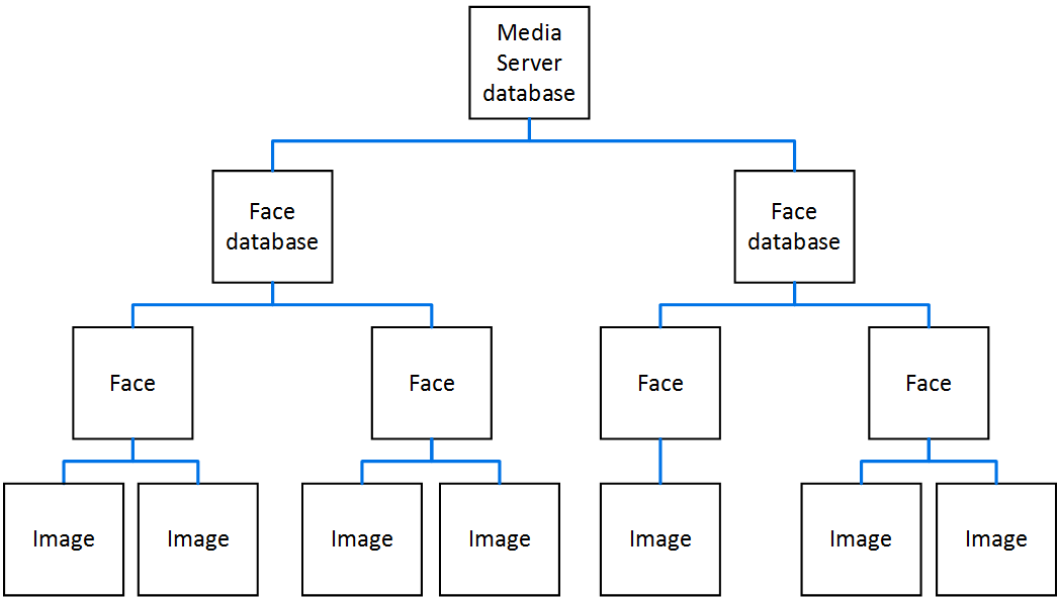
   For more information about the parameters that you can use to configure Face Detection, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 image or 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 Introduction, on page 42.

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. Micro Focus 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.

Micro Focus 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:

- should contain only one face. An image that contains multiple faces is accepted only if one of the faces covers more than double the image area of the others. The largest face is used to train Media Server and the others are ignored.
- 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`	(Set this or `imagepath`, but not both). 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 78.
`imagepath`	(Set this or `imagedata`, but not both). The paths of the training images to add. The paths must be absolute or relative to the Media Server executable file.
`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, to add training images by supplying the image data:

```
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
```

Alternatively, to add training images by supplying their paths:

```
curl http://localhost:14000 -F action=AddFaceImages
                            -F database=Faces
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
                            -F imagepath=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 147.

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`	(Set this or `imagepath`, but not both). 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 78.
`imagepath`	(Set this or `imagedata`, but not both). The paths of the training images to add. The paths must be absolute or relative to the Media Server executable file.
`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\""
```

Alternatively, the following example provides the paths of the training images rather than sending the image data:

```
curl http://localhost:14000 -F action=TrainFace
                            -F database=Faces
                            -F imagepath=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.

   - The `status` element indicates the status of training:
     - `trained` indicates that training was successful.
     - `untrained` indicates that training has not been attempted. Run training for the face using the action `BuildFace`, or run training for all faces that have incomplete training using the action `BuildAllFaces`.
     - `failed` indicates that 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 image using the action `RemoveFaceImages`.
   - The `hasimagedata` element indicates whether the training image is stored in the database. If the value of this element is `false`, the image has been removed from the database by the action `NullFaceImageData`. Images that have been removed and have a status of `untrained`

cannot be trained, so Micro Focus recommends you remove these images with 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 move a face to a different database, use the action `MoveFace`.
- 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*.

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 143.

To recognize faces in media

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=FaceDetect
   Engine2=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. Providing you have sufficient computational resources, Micro Focus recommends setting this parameter to the `DataWithSource` output track from your face detection task. For example, if your face detection task is named `FaceDetect`, set this parameter to `FaceDetect.DataWithSource`. Alternatively, you can use the `ResultWithSource` track produced by your face detection task.
`Database`	(Optional) The database to use for recognizing the detected faces. By default, Media Server uses all available data. Database names are case-sensitive.
`MaxFaces`	The total number of faces that you want to recognize. For example if there are 600 faces in your face database, set `MaxFaces=600`. The value of this parameter determines how many channels are required to run the face recognition task.
`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).

   For example:

   ```
   [FaceRecognize]
   Type=FaceRecognize
   Input=FaceDetect.DataWithSource
   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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 143.

To obtain demographic information for detected faces

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=FaceDetect
   Engine2=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. Providing you have sufficient computational resources, Micro Focus recommends setting this parameter to the `DataWithSource` output track from your face detection task. For example, if your face detection task is named `FaceDetect`, set this parameter to `FaceDetect.DataWithSource`. Alternatively, you can use the `ResultWithSource` track produced by your face detection task.

   For example:

   ```
   [FaceDemographics]
   Type=demographics
   Input=FaceDetect.DataWithSource
   ```

   For more information about these parameters, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 143.

To analyze facial expression

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=FaceDetect
   Engine2=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. Providing you have sufficient computational resources, Micro Focus recommends setting this parameter to the `DataWithSource` output track from your face detection task. For example, if your face detection task is named `FaceDetect`, set this parameter to `FaceDetect.DataWithSource`. Alternatively, you can use the `ResultWithSource` track produced by your face detection task.

   For example:

   ```
   [FaceState]
   Type=facestate
   Input=FaceDetect.DataWithSource
   ```

   For more information about these parameters, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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>
        <percentageInImage>100</percentageInImage>
        <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.
- `percentageInImage` indicates how much of the face is within the image boundary. If a face appears on the edge of an image and is only partially visible, this value will be less than `100`.
- 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 towards the camera (so that `outOfPlaneAngleX` is less than `90`), 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 towards the camera (so that `outOfPlaneAngleX` is less than `90`).

> **TIP:**
> Face detection can return co-ordinates that are negative. For example, in the `region` element the values for `left` and `top` can be negative if a face is detected on the edge of an image. In cases where a face fills the source image, the values for `width` and `height` might also exceed the image dimensions.

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>A N Example</identifier>
      <database>politicians</database>
      <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 154.
- The `identity` element contains information about the recognized face.
  - 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 `confidence` element provides the confidence score for the match.

  The `identity` element can be empty when there are no matches that meet the recognition threshold. You can configure Media Server to output only recognized faces by setting `OutputIdentities=Known`.

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 154.
- 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 154.
- The `expression` element describes the facial expression (`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 value might be `unknown` if the face is only partially visible or the person is not looking at the camera.
- The `spectacles` element specifies whether the person is wearing spectacles. The possible results are `true`, `false`, or `unknown`. The value might be `unknown` if the face is only partially visible or the person is not looking at the camera.

Automatically Enroll Unrecognized Faces

Media Server can automatically enroll unrecognized faces (add them to the training database).

A configuration to enroll unrecognized faces is included in the Media Server installation folder (`configurations/examples/Face/Enroll.cfg`). It contains the following steps:

- Face detection, to detect all faces that appear in the media.
- Face recognition, to determine whether a face is already present in the training database. This task uses the `RecognitionThreshold` parameter to specify the minimum confidence required to successfully recognize a face. The `OutputIdentities` parameter is set to `Unknown` so that the output includes only unknown faces.
- An event stream processing (ESP) task, to discard faces that are in profile or partially outside the image. To achieve the best performance, face recognition should be trained with images that show a complete face with both eyes visible.
- A crop transformation task, to create cropped images that each show a single unrecognized face. Images that contain more than one face cannot be used for training face recognition, so this step is important in case several unrecognized people appear at the same time.
- A rotate transformation task, to rotate the images (if necessary) so that the face is upright.
- An enroll task, to enroll the images of unrecognized faces in the database.
- An output task so that you can see what Media Server has added to the database. To see the images, you could also add an encoding task, or use the action `GetFaceImage` after enrollment is complete.

The following procedure describes how to configure an enroll task.

To automatically enroll faces

1. Open the configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `EngineN` parameter.
3. Create a new configuration section that matches the name of the task and configure Media Server to enroll the images. Use the following configuration parameters:

`Type`	The engine to use. Set this parameter to **`Enroll`**.
`Module`	Set this parameter to **`Face`**.
`Input`	The track that contains the images to enroll.
`Database`	The name of the database to add the faces to. Micro Focus recommends that you add faces to a new database, not the database you use for recognition. An operator can then check the enrolled faces. For example, you should make sure that all of the images added to

	a face represent the same person. After verifying the enrolled faces, you can move them to the correct recognition database.
`PostAction`	Specifies what Media Server should do after enrolling an image in the database. The default value, `build`, trains Media Server to recognize the face.
`PostSyncDatabase`	Specifies whether Media Server should synchronize with the training database after enrolling an image. You might want to use this parameter if you are adding faces to the same database that is used for recognition. If an unrecognized face appears again but you have not trained Media Server and synchronized with the database, the face remains unrecognized. This means that Media Server adds another entry to the database for the same face. If you are enrolling faces in a different database to the one that is used for recognition, there is no need to set this parameter.
`Identifier`	The identifier to use when adding a new face to the database. Micro Focus recommends using the macro `%record.id%`, because this identifier is set by the face detection task and is unique for each detected face.

For example:

```
[EnrollFaces]
Type=Enroll
Module=Face
Input=CropFaces.Output
Database=EnrolledFaces
PostAction=Build
PostSyncDatabase=TRUE
Identifier=%record.id%
```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Face Enrollment Results

The following XML shows a single record that is produced when Media Server enrolls an image of a face.

```
<record>
   ...
   <trackname>EnrollFaces.Result</trackname>
   <EnrollResult>
      <id>608f22bb-06ad-469f-811b-5d6cabd0db98</id>
      <identity>
         <identifier>608f22bb-06ad-469f-811b-5d6cabd0db98</identifier>
         <database>EnrolledFaces</database>
         <imagelabel>f6e5e0f8aa471812fe0e01c4dd35c05d</imagelabel>
```

```
      </identity>
      <error>Failed: no face detected</error>
   </EnrollResult>
</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 enroll engine use the same IDs as the input records.
- The `identity` element provides information about the image that was enrolled in the database.
  - The `database` element provides the name of the database that the image was added to.
  - The `identifier` element provides the identifier of the face that the image was added to.
  - The `imagelabel` element provides the label that was created to identify the enrolled image.
- The `error` element is present only when an error occurs and provides information about the problem.

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.

- To be detected, faces must exceed the minimum size specified by the `MinSize` parameter in your task configuration.
- 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). Micro Focus 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.

- For face recognition, Micro Focus recommends that you configure Media Server to return the top five or ten results and then have a person select the best match from these results. Using face recognition in this way can produce better accuracy than using Media Server alone.

Chapter 12: 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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 376.
`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 `percent`). To specify the position and size of the region in pixels, set this parameter to **`pixel`**.

   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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 the origin of the text. The possible values are:
  - `image` - static text from an image or video.
  - `scroller, left` - text from video of a news ticker, with text scrolling from right to left.
  - `text` - the text originated from a text element in a document.

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.

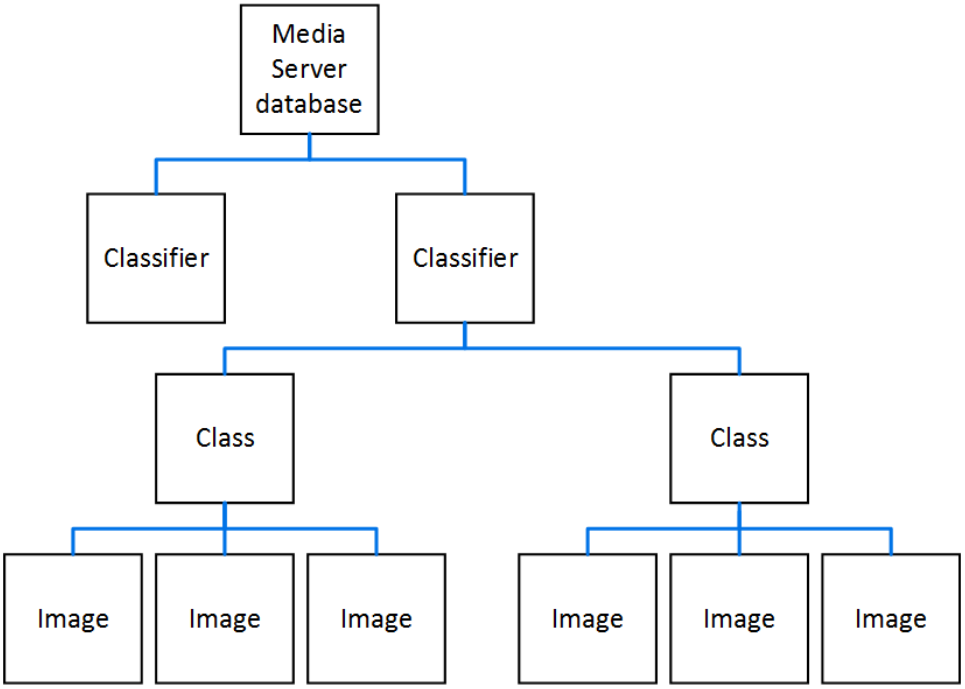
Chapter 13: Image Classification

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 Introduction, on page 42.

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 classes named "car", "truck", and "motorcycle".

A classifier must contain at least two classes. To determine whether an image contains a car or doesn't contain a car, you would need to train a "car" class and also a "not car" class.

To each class you must add training images. Usually around 100 training images per class is sufficient to obtain good performance. For information about choosing suitable training images, see Training Requirements, below.

Training Requirements

Media Server image classification uses Convolutional Neural Network (CNN) classifiers. A CNN classifier usually produces more accurate results than other types of classifier, 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 three hours to complete on a CPU or five minutes to complete with a GPU. Micro Focus 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 on a CPU is likely to take weeks.

To find the optimum number of iterations, Micro Focus 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 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.

The performance of classification is generally better if:

- the classifier contains only a few classes (but it must contain at least two classes).
- the 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 classes are trained with many images. Usually around 100 images are sufficient to train a 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 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, Micro Focus recommends cropping the image. If you crop an image, leave a gap around the object of at least 16 pixels.

Create a Classifier

Image classification classifies images or video frames into the classes that are defined by a 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 number of training iterations) 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=iterations
                               -F value=1000
   ```

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
`iterations`	The number of iterations to perform when training a neural network classifier. For information about how to set this training option, see Training Requirements, on page 169.	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 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 class to. The classifier must already exist. To create a classifier, see Create a Classifier, on the previous page.

   `identifier` (Optional) A unique identifier for the class (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.

   `imagedata` (Set this or `imagepath`, but not both). 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 78.

   `imagepath` (Set this or `imagedata`, but not both). The paths of the training images to add to the class. The paths must be absolute or relative to the Media Server executable file.

   `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,...
```

   Alternatively, the following example provides the paths of the training images rather than sending the image data:

```
curl http://localhost:14000 -F action=createclass
                            -F classifier=vehicles
                            -F identifier=cars
                            -F imagepath=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 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 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>iterations</key>
```

```
          <value>2000</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 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 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 classes in the specified classifier.

If you set the action parameter `imagelabels` to **`true`**, Media Server returns the labels of images associated with each class.

If you set the action parameter `metadata` to **`true`**, Media Server returns the metadata you have added to each class.

Update or Remove Classes and Classifiers

To update or remove a class use the following actions:

- To add additional training images to a class, use the action `AddClassImages`. After adding images to a class you must retrain the classifier using the action `BuildClassifier`. To confirm that training was successful, use the action `ListClassifiers` (see List Classifiers and Classes, on the previous page).
- To remove training images from a class, use the action `RemoveClassImages`. After removing images from a class you must retrain the classifier using the action `BuildClassifier`. To confirm that training was successful, use the action `ListClassifiers` (see List Classifiers and Classes, on the previous page).
- To change the label of an image that you added to a class, use the action `RelabelClassImage`.
- To move an image from one class to another, for example if you add an image to the wrong class, use the action `MoveClassImage`.
- To add, remove, or update custom metadata for a class, use the actions `AddClassMetadata`, `RemoveClassMetadata`, and `UpdateClassMetadata`.

- To rename a class, use the action `RenameClass`.
- To remove a 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 modify the training options for a classifier, use the actions `SetClassifierTrainingOption` and `UnsetClassifierTrainingOption`.
- 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*.

Import a Classifier

Micro Focus 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 385.

To import a classifier

1. Go to the MySupport portal and download the classifier to your Media Server machine. When you download the classifier, ensure the version of the classifier matches the version of Media Server that you are using.
2. Import the training data by running the action `ImportClassifer`, for example if you downloaded the file to the folder `pretrained`, in the Media Server installation folder:

   ```
   /action=ImportClassifier&classifier=imagenet
                          &classifierpath=./pretrained/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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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 **`ImageClassification`**.
`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 images 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=ImageClassification
   Classifier=Vehicles
   ClassificationThreshold=15
   ```

   For more information about these parameters, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Classification Results

The following XML shows a single record produced by image classification.

```
<output>
  <record>
    ...
    <trackname>Classification.Result</trackname>
```

```
    <ImageClassificationResult>
       <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>
    </ImageClassificationResult>
  </record>
</output>
```

The record contains the following information:

- The `id` element provides a unique identifier for the result.
- The `classification` element represents a match between the image and a class in your chosen classifier.
  - The `identifier` element provides the identifier of the 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 class in the training database. If there is no metadata associated with the class, this element is omitted.
- The `classifier` element provides the name of your chosen classifier.

Chapter 14: Object Class Recognition

Object class recognition 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 class recognition 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.

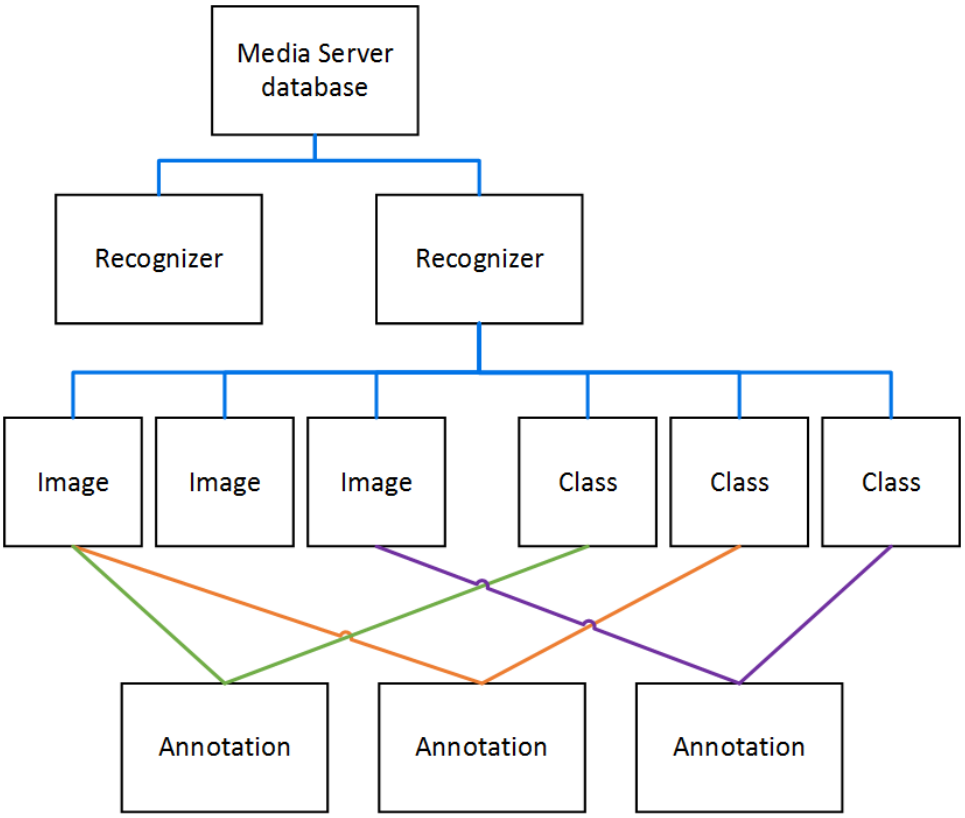
Object class recognition differs from image classification because it returns the position for every recognized object. Object class recognition can find multiple objects within the analyzed image or region, whereas classification returns a single classification result and no position information. For example, using object class recognition 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 Recognize Objects

Before using object class recognition, you must either import a recognizer that is provided by Micro Focus, or train your own recognizer. The recognizers provided by Micro Focus are pre-trained, so you do not need to perform any training. For information about the recognizers that are available, see Pre-Trained Object Class Recognizers, on page 386.

> **NOTE:**
> To train your own recognizers you must have a supported GPU and enable GPU acceleration.

The following diagram shows how Media Server stores information for object class recognition:



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 Training Database, on page 42.

The Media Server database can contain any number of recognizers. An object class recognition task uses a single recognizer and finds instances of objects that are described by classes in that recognizer. For example, if you are processing video captured by a CCTV camera, you could use the `RoadScene` recognizer which contains classes for finding people, cars, and vans.

To train object class recognition you add your training images and chosen classes to the recognizer. You then add *annotations* to the images. An annotation identifies a region of a training image that contains an example object, and the class that the object belongs to. Media Server can then use that region of the image to train the recognizer. You can add multiple annotations to the same image, which is useful if an image contains more than one example object.

Create and Train a Recognizer

To create and train an object class recognizer, follow these steps. For more information about the actions used in this section, refer to the *Media Server Reference*.

To create and train a recognizer

1. Start by creating a new recognizer. For example:

```
curl http://localhost:14000 -F action=CreateObjectClassRecognizer
                            -F recognizer=vehicles
```

2. Add your chosen object classes to the recognizer.

   The following example adds a new class named "car" to the "vehicles" recognizer.

```
curl http://localhost:14000 -F action=CreateObjectClass
                            -F recognizer=vehicles
                            -F identifier=car
```

3. Add images to the recognizer that contain example objects for training.

   The following example adds two images to the "vehicles" recognizer, and labels them "car42" and "bike19":

```
curl http://localhost:14000 -F action=AddObjectClassImages
                            -F recognizer=vehicles
                            -F imagedata=@car42.jpg,bike19.jpg
                            -F imagelabels=car42,bike19
```

4. Annotate each training image. An annotation identifies a region of a training image that contains an object, and the class that the object belongs to. Media Server can then use that region of the image to train the recognizer. You can add multiple annotations to the same image, if the image contains multiple objects.

   The following example adds an annotation to the image "car42", associating a region covering the center 25% of the image with the "car" object class. The region is specified by the `boxes` parameter, which is a comma-separated list of values in the form `(`*`Left`*`,`*`top`*`,`*`width`*`,`*`height`*`)`. *`Left`* specifies the distance from the left side of the image to the left side of the region. *`Top`* specifies the distance from the top of the image to the top of the region. *`Width`* and *`Height`* specify the width and height of the region. The values are in pixels, unless you set `regionaspercentage=true` (in which case specify *`Left`* and *`Width`* as a percentage of the image width and *`Top`* and *`Height`* as a percentage of the image height).

```
curl http://localhost:14000 -F action=AddObjectClassAnnotations
                            -F recognizer=vehicles
                            -F imagelabel=car42
                            -F identifiers=car
                            -F boxes=(25,25,50,50)
                            -F regionaspercentage=true
```

   > **TIP:**
   > To obtain a list of training images that have been added to the recognizer but have not been annotated, use the action `ListUnannotatedObjectClassImages`.

5. Train Media Server by building the recognizer. For example:

```
curl http://localhost:14000 -F action=BuildObjectClassRecognizer
                            -F recognizer=vehicles
```

Import a Recognizer

Micro Focus may provide recognizers that you can use with Media Server to run object class recognition. These 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 recognizers, see Pre-Trained Object Class Recognizers, on page 386.

To import a recognizer

1. Go to the MySupport portal and download the recognizer to your Media Server machine. When you download the recognizer, ensure the version matches the version of Media Server that you are using.
2. Import the training data by running the action `ImportObjectClassRecognizer`, for example if you downloaded the file to the folder `pretrained`, in the Media Server installation folder:

   ```
   /action=ImportObjectClassRecognizer&recognizer=roadscene
           &recognizerpath=./pretrained/roadscene.dat
   ```

   where,

`recognizer`	The name to give to the imported recognizer.
`recognizerpath`	The path of the recognizer data file on disk
`recognizerdata`	The recognizer data (you can set this as an alternative to `recognizerpath`).

   Media Server imports the data. You can now run object class recognition.

Recognize Objects

To recognize objects

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=ObjectClassRecognition
   ```

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 **`ObjectClassRecognition`**.

`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.
`Recognizer`	The recognizer to use.
`DetectionThreshold`	(Optional) The minimum confidence score necessary for Media Server to output a result. Any result with a confidence score below the threshold is discarded.

For example:

```
[ObjectClassRecognition]
Type=ObjectClassRecognition
Recognizer=roadscene
DetectionThreshold=30
```

For more information about these parameters, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Object Class Recognition Results

The following XML shows two records produced by object class recognition.

```
<output>
  <record>
    ...
    <trackname>ObjectClassRecognition.Result</trackname>
    <ObjectClassRecognitionResult>
      <id>d5a249f7-3207-4a41-9a23-0b8ac4aafbf6</id>
      <classification>
        <identifier>van</identifier>
        <confidence>100</confidence>
      </classification>
      <recognizer>roadscene</recognizer>
      <region>
        <left>31</left>
        <top>54</top>
        <width>206</width>
        <height>159</height>
      </region>
    </ObjectClassRecognitionResult>
  </record>
  <record>
    ...
    <trackname>ObjectClassRecognition.Result</trackname>
    <ObjectClassRecognitionResult>
      <id>d5a249f7-3207-4a41-9a23-0b8ac4aafbf7</id>
```

```
      <classification>
        <identifier>person</identifier>
        <confidence>99.94</confidence>
      </classification>
      <recognizer>roadscene</recognizer>
      <region>
        <left>243</left>
        <top>33</top>
        <width>96</width>
        <height>182</height>
      </region>
    </ObjectClassRecognitionResult>
  </record>
</output>
```

Each record contains the following information:

- The `id` element provides a unique identifier for the recognized object.
- The `classification` element provides information about the recognized object.
  - The `identifier` element provides the identifier of the class that resulted in the object being recognized.
  - The `confidence` element provides the confidence score (from 0 to 100).
- The `recognizer` element provides the name of the object class recognizer that was used.
- The `region` element provides the location of the object in the image or video frame.

Chapter 15: Object Recognition

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

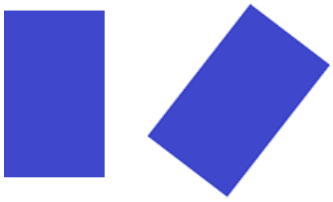
For example, a photograph or video frame that shows product packaging.

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)		✓

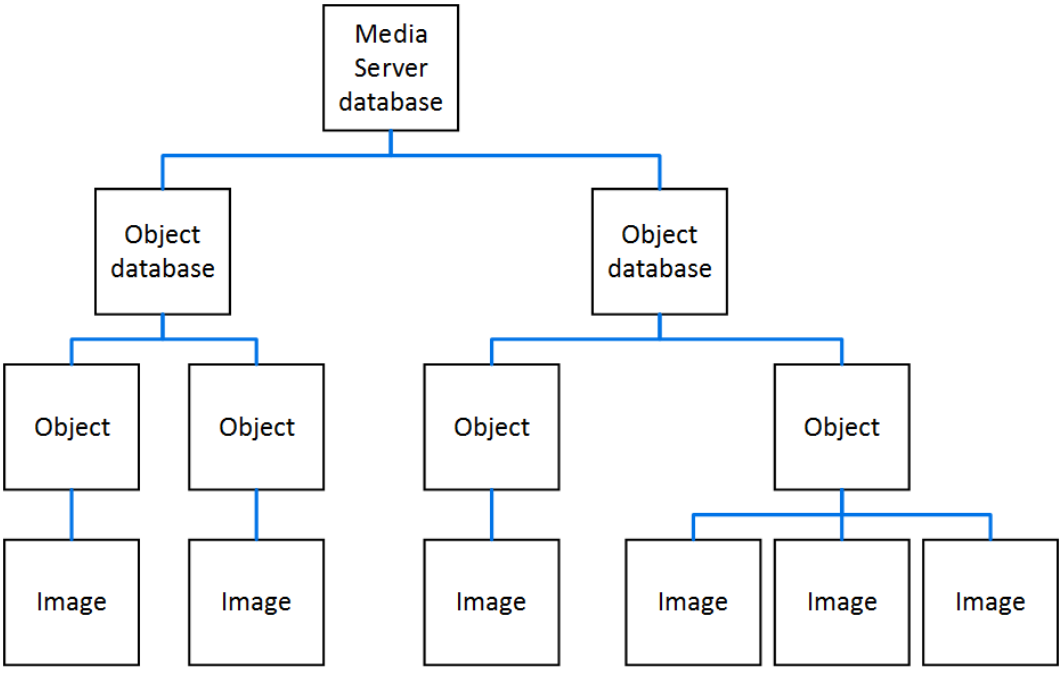
2-D object on a surface that is not flat		✗

TIP:
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 Introduction, on page 42.

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. Micro Focus 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 a 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 189.

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`	(Set this or `imagepath`, but not both). 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 78.
`imagepath`	(Set this or `imagedata`, but not both). The paths of the training images to add. The paths must be absolute or relative to the Media Server executable file.
`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, to add a training image by supplying the image data:

```
curl http://localhost:14000 -F action=AddObjectImages
                            -F database=CompanyLogos
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
                            -F imagedata=@logo1.png
```

Alternatively, to add a training image by supplying its path:

```
curl http://localhost:14000 -F action=AddObjectImages
                            -F database=CompanyLogos
                            -F identifier=6600dc0f9dd72d0cb55589e8f1d28b62
                            -F imagepath=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 191.
`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 187.

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`	(Set this or `imagepath`, but not both). 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 78.
`imagepath`	(Set this or `imagedata`, but not both). The paths of the training images to add. The paths must be absolute or relative to the Media Server executable file.
`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
```

Alternatively, the following example provides the path of the training image rather than sending the image data:

```
curl http://localhost:14000 -F action=TrainObject
                            -F database=CompanyLogos
                            -F imagepath=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 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`.	true, false	false
`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 187.

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.

- The `status` element indicates the status of training:
  - `trained` indicates that training was successful.
  - `untrained` indicates that training has not been attempted. Run training for the object using the action `BuildObject`, or run training for all objects that have incomplete training using the action `BuildAllObjects`.
  - `failed` indicates that Media Server could not use the image for training. Remove the failed image using the action `RemoveObjectImages`.
- The `hasimagedata` element indicates whether the training image is stored in the database. If the value of this element is `false`, the image has been removed from the database by the action `NullObjectImageData`. Images that have been removed and have a status of `untrained` cannot be trained, so Micro Focus recommends you remove these images with the action `RemoveObjectImages`.

Update or Remove Objects and Databases

To update or remove an object use the following actions:

- To complete training for all objects that exist in the Media Server database but have incomplete training, use the action `BuildAllObjects`. To confirm that training was successful, use the action `ListObjects` (see List the Objects in a Database, on the previous page).
- 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 move an object to a different database, use the action `MoveObject`.
- 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*.

Recognize Objects

To recognize objects, configure an object analysis task by following these steps.

To recognize objects

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=ObjectRecognition
   ```

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 **`ObjectRecognition`**.
`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:

```
[ObjectRecognition]
Type=ObjectRecognition
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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Object Recognition Results

The following XML shows a single record produced by object recognition.

```
<output>
  <record>
    ...
    <trackname>object.Result</trackname>
    <ObjectRecognitionResult>
      <id>ed1f3af7-9f00-434f-8bc4-a328ff4c67fc</id>
      <identity>
        <identifier>MicroFocus</identifier>
        <database>logos</database>
        <confidence>100</confidence>
        <metadata>
          <item>
            <key>CompanyName</key>
            <value>MicroFocus</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>
    </ObjectRecognitionResult>
  </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 `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). Micro Focus 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 16: Text Detection

Media Server can detect regions, in images and video, that contain text.

Introduction

Media Server can detect regions, in images and video, that contain text.

You can also use text detection to identify vehicles (by finding the text on the vehicle's number plate), and then identify the vehicle's make, model, and color (see Vehicle Make and Model Recognition, on page 203). You should only use text detection for this purpose if you cannot use number plate recognition, for example when the number plates exhibit out of plane rotation, because text detection identifies all text in a frame rather than just the number plates.

Text detection does not read the text. If you want to detect and recognize text, you can use OCR, or run text detection to locate regions that contain text and then run OCR on those regions. In cases where the media contains small amounts of text within a scene (and you would use `OCRMode=scene`), using text detection to find the text and then running OCR can provide better performance than running OCR on the entire image or video frame. Where there is a large amount of text, for example an image of a document, using OCR (with `OCRMode=Document`) is the best approach.

In some cases text detection might be able to detect text that cannot be read by OCR or number plate recognition, for example due to out-of-plane skew. You can use text detection to ensure that the text is captured, to be read and evaluated by a person. This might be useful for saving images or video for evidential purposes, for example.

Set up Text Detection

To detect regions that contain text

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=DetectText
   ```

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 **`TextDetection`**.
`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.
`DetectionThreshold`	(Optional) The minimum confidence score required for Media Server to detect text.
`NumParallel`	(Optional) The maximum number of CPU threads to use for analysis. In most cases this specifies the number of video frames to analyze concurrently. This parameter has no effect on single images (Media Server always uses a single thread on images).

For example:

```
[DetectText]
Type=TextDetection
DetectionThreshold=55
```

For more information about the parameters you can use to customize text detection, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Text Detection Results

The following XML shows a single record produced by text detection.

```
<record>
   ...
   <trackname>DetectText.Result</trackname>
   <TextDetectionResult>
      <id>aa056c5e-3cea-4cfa-a090-713384270424</id>
      <region>
         <left>264</left>
         <top>280</top>
         <width>106</width>
         <height>18</height>
      </region>
      <confidence>100</confidence>
   </TextDetectionResult>
</record>
```

The record contains the following information:

- The `id` element provides a unique identifier for each example of detected text. The text detection engine does not track text across video frames, so if you process video and the same text appears in consecutive frames there will be records (with different identifiers) in the result track for each frame. There might be multiple records per frame if text is detected in more than one region.

- The `region` element describes the position of the detected text in the image or video frame (as a rectangle). `left` provides the number of pixels between the left side of the image and the left side of the region. `top` provides the number of pixels between the top of the image and the top of the region. `width` and `height` provide the width and height of the region.
- The `confidence` element describes the confidence score for the detection, from 0 to 100, where higher values represent greater confidence.

Example Configuration

The following is an example configuration that demonstrates how to detect text in ingested images. The configuration includes a transformation task to draw the region(s) and an encoding task to output the resulting image(s).

```
[Session]
Engine0=Ingest
Engine1=DetectText
Engine2=DrawRegion
Engine3=EncodeImage
Engine4=Output

[Ingest]
Type=Image

[DetectText]
Type=TextDetection
DetectionThreshold=50

[DrawRegion]
Type=Draw
Input=DetectText.ResultWithSource
Color=Red
Thickness=3

[EncodeImage]
Type=ImageEncoder
ImageInput=DrawRegion.Output
OutputPath=output/%session.token%/%segment.sequence%-%source.filename%

[Output]
Type=Response
Input=DetectText.Result
```

Chapter 17: Number Plate Recognition

Media Server can detect and read the number plates of vehicles in a scene. Number plate recognition has many applications; you can detect stolen and uninsured vehicles, monitor the length of stay for vehicles in car parks, and provide a drive-off deterrent at petrol filling stations.

Requirements for ANPR

For reliable number plate detection and recognition, ensure that your system meets the following requirements.

Camera	Micro Focus 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.

2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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`	(Set this or `LocationWithPriorities`) A comma-separated list of ISO-3166 codes to specify the locations for which you want to recognize number plates. Specify the primary location first. For example, if you are recognizing number plates in France but also want to recognize plates from Germany and Belgium, set this parameter to `Location=FR,DE,BE`.
`LocationWithPriorities`	(Set this or `Location`) A comma-separated list of ISO-3166 codes to specify the locations for which you want to recognize number plates, and their relative priorities. For example, if you are recognizing number plates in France but are near the border and also want to recognize plates from Germany and Belgium, you might set `LocationWithPriorities=fr:1.0,de:0.1,be:0.01`. This instructs Media Server that French, German, and Belgian number plates are expected, but German plates are 10 times less likely to be seen than French plates, and Belgian number plates are 100 times less likely to be seen than French plates.
`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.
`MinValidScore`	(Optional) The average character score required for a number plate to be recognized.

   For example:

   ```
   [ANPR]
   Type=numberplate
   ```

```
Location=GB
RegionUnit=percent
Region=20,10,60,90
Sensitivity=14
```

For more information about the parameters that customize number plate recognition, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 18: Vehicle Make and Model Recognition

Media Server can identify the make, model, and color of vehicles that are detected in a scene.

Introduction

Media Server can recognize the make, model, and color of vehicles that are detected in a scene.

Vehicle make and model recognition can help law enforcement identify stolen vehicles. If you also run number plate recognition, you can compare the make and model detected by Media Server to a database and identify vehicles that have false number plates.

To recognize the make and model of a vehicle, you must first identify the vehicle within the scene. You can do this by running text detection or number plate recognition to detect the vehicle's number plate. The vehicle model analysis engine requires as input either the `DataWithSource` or `ResultWithSource` track from a number plate analysis task, or the `ResultWithSource` track from a text detection task.

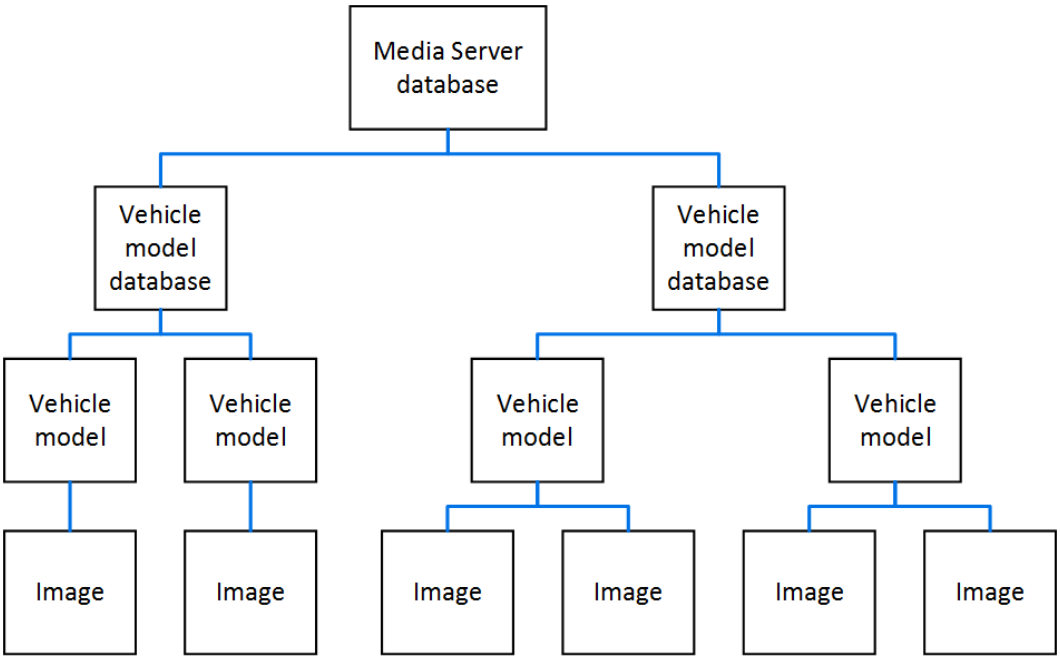
To recognize the color of a vehicle, you must first identify the vehicle make.

Train Media Server to Recognize Vehicle Models

Vehicle recognition identifies both the make and model of a vehicle detected by number plate recognition.

Media Server has been pre-trained to identify the make of vehicles, but you must train Media Server to recognize vehicle models by providing images of those models.

The following diagram shows how Media Server stores information about vehicle models.



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 Introduction, on page 42.

You can organize vehicle models into databases. These databases are for organizational purposes. For example, you might have a database for European vehicle models, and a database for American vehicle models. When you run vehicle recognition, you must choose the database to use.

A database can contain any number of vehicle models. You must associate a vehicle make with each model. After Media Server identifies the make of a vehicle it can then perform recognition against a much smaller number of models.

To each vehicle model you must add at least one training image. Vehicle manufacturers update the design of their models over time, and might offer vehicles in different configurations. You can choose whether to consider these the same model or train different models.

Obtain Images for Training

Vehicle model recognition performs best when vehicles are moving towards the camera, and the front of the vehicle is visible in the video. Ideally your cameras should be positioned above the lane(s) of traffic being monitored, and not at the roadside. Cameras that are positioned directly above the traffic are better for recognition because they capture images where the vehicles approach head-on.

Usually three to five training images are sufficient to train each model. You should supply images from several angles (for example, head-on, 15 degrees left/right of center, and 30 degrees left/right of center). It can help to add a set of images at a smaller size and a set at a larger size, in case vehicles are detected at different distances.

The training images that you use to train a single model should be different from each other, such that each image adds new information to the model. Adding many almost identical images to a single model is unlikely to improve accuracy and increases the time required for processing.

You can use Media Server to obtain training images. Run vehicle make and model recognition on a sample video, crop the video frames to the region identified by the vehicle model analysis engine, and encode the cropped images using the image encoder. Media Server includes an example configuration, `configurations/examples/VehicleModel/OutputModelPatches.cfg`, that performs these steps. You can then use a selection of the encoded images for training.

In some cases you might need to produce the training images manually. Media Server might not crop frames successfully for some types of vehicles, including larger vehicles such as trucks. If you are training Media Server to recognize a brand new model and the prototype vehicle does not have a number plate, Media Server cannot produce training images because the number plate is used to identify the position of the vehicle in the image.

List the Supported Vehicle Makes

Media Server has been pre-trained to identify the make of many vehicles. To see the list of supported vehicle makes, use the action `ListVehicleMakes`.

To list the supported vehicle makes

- Run the action `ListVehicleMakes`, for example:

  ```
  curl http://localhost:14000 -F action=ListVehicleMakes
  ```

  Media Server returns the list of supported makes.

If you want to recognize a vehicle model that is not from one of the supported makes, you can train the vehicle model using the make **`Unknown`**.

To request support for a vehicle make, first obtain training images. Media Server includes a sample configuration, `configurations/examples/VehicleModel/OutputMakePatches.cfg`, that encodes suitable images. The configuration saves images into folders which match the name of an existing vehicle make, or "unknown". The images for an unknown make might be classified as "unknown", but depending on the recognition threshold and the similarity with existing vehicle makes, they might also be classified, incorrectly, as an existing vehicle make. After obtaining training images, contact Micro Focus IDOL Technical Support.

Create a Database to Contain Vehicle Models

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

To create a database to contain vehicle models

- Use the `CreateVehicleModelDatabase` action, with the `database` parameter:

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

  For example:

  ```
  curl http://localhost:14000 -F action=CreateVehicleModelDatabase
                              -F database=Cars
  ```

Add a Vehicle Model to a Database

To add a vehicle model to a database of vehicle models, you can:

- **Perform training in separate steps**. Create a new (empty) model, add training images, and then train Media Server, using separate actions. You can also add metadata to the vehicle model, but this is an optional step.
- **Perform training in a single action**. Add and train a new vehicle model with a single action. You might use this approach when all of the required information is available and ready to be added to the database. When adding a vehicle model using a single action, either all of the training steps are successful or nothing is added to the database.

Add a Vehicle Model to a Database (Using Separate Steps)

This section describes how to create a new (empty) vehicle model, add training images, and then train Media Server, using separate actions. You can also add metadata to the vehicle model, but this is an optional step.

Alternatively, you can train Media Server to recognize a vehicle model by sending a single action. To do this, see Add a Vehicle Model to a Database (Using a Single Action), on page 208.

To add a vehicle model to a database (using separate steps)

1. Add a new vehicle model using the `NewVehicleModel` action. Set the following parameters:

`database`	The name of the database to add the vehicle model to. The database must already exist.
`identifier`	(Optional) A unique identifier for the vehicle model (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.
`vehiclemake`	The vehicle manufacturer. You must set this to one of the makes returned by the action ListVehicleMakes, or **`Unknown`**.

   For example:

   ```
   curl http://localhost:14000 -F action=NewVehicleModel
                               -F database=Cars
                               -F identifier=FordFocus
                               -F vehiclemake=Ford
   ```

   Media Server adds the vehicle model to the database and returns the identifier.

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

`database`	The name of the database that contains the vehicle model.

`identifier`	The identifier for the vehicle model, returned by the `NewVehicleModel` action.
`imagedata`	(Set this or `imagepath`, but not both). 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 78.
`imagepath`	(Set this or `imagedata`, but not both). The paths of the training images to add. The paths must be absolute or relative to the Media Server executable file.
`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, to add a training image by supplying the image data:

```
curl http://localhost:14000 -F action=AddVehicleModelImages
                            -F database=Cars
                            -F identifier=FordFocus
                            -F imagedata=@ford-focus.png
```

Alternatively, to add a training image by supplying its path:

```
curl http://localhost:14000 -F action=AddVehicleModelImages
                            -F database=Cars
                            -F identifier=FordFocus
                            -F imagepath=./images/ford-focus.png
```

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

`database`	The name of the database that contains the vehicle model.
`identifier`	The identifier for the vehicle model, returned by the `NewVehicleModel` 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=AddVehicleModelMetadata
                            -F database=Cars
                            -F identifier=FordFocus
                            -F key=type
                            -F value=hatchback
```

4. Complete the training for the vehicle model using the `BuildVehicleModel` action. Set the following parameters:

`database`	The name of the database that contains the vehicle model.
`identifier`	The identifier for the vehicle model, returned by the `NewVehicleModel` action.

For example:

```
curl http://localhost:14000 -F action=BuildVehicleModel
                            -F database=Cars
                            -F identifier=FordFocus
```

Add a Vehicle Model to a Database (Using a Single Action)

You can train Media Server to recognize a vehicle model by sending a single action (`TrainVehicleModel`).

Running this action is equivalent to running the following actions in the following order:

- `NewVehicleModel`
- `AddVehicleModelImages`
- `AddVehicleModelMetadata` (optional)
- `BuildVehicleModel`

The `TrainVehicleModel` 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. For more information about how to do this, see Add a Vehicle Model to a Database (Using Separate Steps), on page 206.

To add a vehicle model to a database (using a single action)

- Add a vehicle model using the `TrainVehicleModel` action. Set the following parameters:

`database`	The name of the database to add the vehicle model to. The database must already exist.
`identifier`	(Optional) A unique identifier for the model (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.
`imagedata`	(Set this or `imagepath`, but not both). 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 78.
`imagepath`	(Set this or `imagedata`, but not both). The paths of the training images to add. The paths must be absolute or relative to the Media Server executable file.
`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 vehicle model. 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 (\).
`vehiclemake`	The vehicle manufacturer. You must set this to one of the makes returned by the action ListVehicleMakes, or **`Unknown`**.

For example:

```
curl http://localhost:14000 -F action=TrainVehicleModel
                            -F database=Cars
                            -F identifier=FordFocus
                            -F vehiclemake=Ford
                            -F imagedata=@ford-focus.png
                            -F metadata=type:hatchback
```

Alternatively, the following example provides the path of the training image rather than sending the image data:

```
curl http://localhost:14000 -F action=TrainVehicleModel
                            -F database=Cars
                            -F identifier=FordFocus
                            -F vehiclemake=Ford
                            -F imagepath=./images/ford-focus.png
```

```
-F metadata=type:hatchback
```

Media Server adds the vehicle model to the database and returns the identifier.

List the Vehicle Models in a Database

To list the vehicle models that you have added to a database, and check whether training was successful, use the following procedure.

To list the vehicle models in a database

1. (Optional) First list the databases that have been created to store vehicle models. Use the action `ListVehicleModelDatabases`:

   ```
   http://localhost:14000/action=ListVehicleModelDatabases
   ```

   Media Server returns a list of databases that you have created.

2. List the vehicle models that exist in one of the databases. Use the action `ListVehicleModels`, for example:

   ```
   http://localhost:14000/action=ListVehicleModels&database=Cars
                                                  &metadata=true
                                                  &imagestatus=true
   ```

   Media Server returns a list of vehicle models in the specified database, and the number of training images associated with each model.

   If you set the action parameter `metadata` to **`true`**, Media Server returns the metadata you have added to each vehicle model.

   If you set the action parameter `imagestatus` to **`true`**, Media Server returns the status of each training image associated with each vehicle model.

   - The `status` element indicates the status of training:
     - `trained` indicates that training was successful.
     - `untrained` indicates that training has not been attempted. Run training for the vehicle model using the action `BuildVehicleModel`, or run training for all vehicle models that have incomplete training using the action `BuildAllVehicleModels`.
     - `failed` indicates that Media Server could not use the image for training. Remove the failed image using the action `RemoveVehicleModelImages`.
   - The `hasimagedata` element indicates whether the training image is stored in the database. If the value of this element is `false`, the image has been removed from the database by the action `NullVehicleModelImageData`. Images that have been removed and have a status of `untrained` cannot be trained, so Micro Focus recommends you remove these images with the action `RemoveVehicleModelImages`.

Update or Remove Vehicle Models and Databases

To update or remove a vehicle model use the following actions:

- To complete training for all vehicle models that exist in the Media Server database but have incomplete training, use the action `BuildAllVehicleModels`. To confirm that training was successful, use the action `ListVehicleModels` (see List the Vehicle Models in a Database, on the previous page).
- To add additional training images to a vehicle model, use the action `AddVehicleModelImages`. Media Server does not use the new images for recognition until you run the action `BuildVehicleModel`. To confirm that training was successful, use the action `ListVehicleModels` (see List the Vehicle Models in a Database, on the previous page).
- To remove a training image from a vehicle model, use the action `RemoveVehicleModelImages`.
- To modify the vehicle make that is associated with a vehicle model, use the action `SetVehicleMake`.
- To change the label of an image that you added to a vehicle model, use the action `RelabelVehicleModelImage`.
- To move an image from one vehicle model to another, for example if you add an image to the wrong model, use the action `MoveVehicleModelImage`.
- To add, remove, or update custom metadata for a vehicle model, use the actions `AddVehicleModelMetadata`, `RemoveVehicleModelMetadata`, and `UpdateVehicleModelMetadata`.
- To change the identifier of a vehicle model you already added to a database, use the action `RenameVehicleModel`.
- To move a vehicle model to a different database, use the action `MoveVehicleModel`.
- To remove a vehicle model from a database and discard all associated images and metadata, use the action `RemoveVehicleModel`.

To update or remove vehicle model databases, use the following actions:

- To rename a vehicle model database, use the action `RenameVehicleModelDatabase`.
- To delete a vehicle model database and all of the information that it contains, use the action `RemoveVehicleModelDatabase`.

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

Recognize Vehicles (Make and Model)

To recognize the make and model of vehicles detected by number plate recognition, use the following procedure.

To recognize vehicles

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=ANPR
   Engine2=VehicleMakeModel
   ```

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. Micro Focus recommends that you use the `DataWithSource` or `ResultWithSource` track from a number plate analysis task, but you can also use the `ResultWithSource` track from a text detection task.
`Database`	The name of the vehicle model database to use to identify vehicles. For information about creating this database, see Obtain Images for Training, on page 204.
`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:

   ```
   [VehicleMakeModel]
   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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Vehicle Make and Model Recognition Results

The following XML shows a single record produced by vehicle make and model recognition.

```
<record>
  ...
  <trackname>VehicleMakeModel.Result</trackname>
  <VehicleModelResult>
    <id>e2fb50f1-a71a-46e0-8af6-c5e7e5ba3afc</id>
    <grillepatch>
      <point>
        <x>168</x>
        <y>172</y>
      </point>
      <point>
        <x>473</x>
        <y>172</y>
      </point>
      <point>
        <x>473</x>
        <y>280</y>
      </point>
      <point>
        <x>168</x>
        <y>280</y>
      </point>
    </grillepatch>
    <identity>
      <identifier>FocusRS</identifier>
      <database>Cars</database>
      <confidence>77</confidence>
      <metadata>
        <item>
          <key>type</key>
          <value>hatchback</value>
        </item>
      </metadata>
    </identity>
    <grillepercentage>100</grillepercentage>
    <vehiclemake>Ford</vehiclemake>
    <vehiclemakescore>100</vehiclemakescore>
  </VehicleModelResult>
</record>
```

The record contains the following information:

- The `id` element provides a unique identifier for the recognized number plate. The number plate recognition engine issues an ID for each detected appearance of a number plate, and the records

output from the vehicle make/model recognition engine use the same IDs as the input records.

- The `grillepatch` element describes the region of the input image that was used for recognizing the vehicle make and model.
- The `identity` element represents a match between the ingested media and a vehicle model in your training database. This element is present only when you train Media Server to recognize vehicle models, you set the `Database` parameter when you configure the task, and when a match is found.
  - The `identifier` element provides the identifier of the vehicle model that was detected in the ingested media.
  - The `database` element provides the name of the database in which the vehicle model exists.
  - The `confidence` element provides the confidence score for the match (from 0 to 100, where 100 is maximum confidence).
  - The `metadata` element provides metadata that you associated with the vehicle model when you trained Media Server. If there is no metadata in the training database, this element is omitted.
- The `vehiclemake` element provides the identified vehicle make.
- The `vehiclemakescore` element provides a confidence score for vehicle make recognition, from 0 to 100, where 100 indicates maximum confidence.

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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=ANPR
   Engine2=VehicleModel
   Engine3=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.
`ColorDictionary`	(Optional) To match vehicle colors against the colors that are defined in a dictionary, specify the path to a dictionary file.
`ColorThreshold`	(Optional) The analysis task discards colors that do not make up a significant proportion of the region (as specified by this parameter).
`ColorSpace`	(Optional) The color space in which the results are provided (**`RGB`**, **`YCbCr`**, **`HSL`**, **`HSV`**, **`CIELAB`**).

For example:

```
[VehicleColor]
Type=ColorCluster
Input=VehicleModel.ColorRegionWithSource
RestrictToInputRegion=TRUE
ColorDictionary=./colorcluster/carcolors.dat
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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 19: Clothing Color Analysis

Media Server can locate the region of an image or video frame that contains 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.

Introduction

To locate the region containing a person's clothing you must first identify people. You can do this by:

- configuring face detection to look for faces
- configuring object class recognition to recognize people with the pre-trained person recognizer.

Find Clothing

Clothing analysis provides the location of the clothing for a person who has been identified by face detection or object class recognition.

To determine the location of clothing

1. Open the configuration file in which you configured your face detection or object class recognition task for identifying people.
2. In the `[Session]` section, add a new analysis task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=FaceDetect
   Engine2=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 on which you want to run analysis. You can specify: • the `DataWithSource` output track from a face detection task. • the `ResultWithSource` output track from any face analysis task.

- the `ResultWithSource` output track from an object class recognition task, when that task runs detection using the pre-trained person recognizer.

`ClothingMode` (Optional) A comma-separated list of regions to identify. You can use the values `Full` (clothing covering the full body), `Upper` (upper body), and `Lower` (lower body). By default, clothing analysis only identifies the location of the clothing covering the full body.

For example:

```
[Clothing]
Type=Clothing
Input=FaceDetect.ResultWithSource
ClothingMode=Upper,Lower
```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 Color Clustering, on page 230. 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 `[Session]` section, add a new analysis task (for color analysis) by setting the `EngineN` parameter. You can give the task any name, for example:

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=Clothing
Engine3=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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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>
     <regiontype>Full</regiontype>
   </ClothingResult>
</record>
```

The record contains the following information:

- The `id` element contains an identifier which is the same as that in the input record. This means that if the person was identified by face detection the identifier matches the ID for the detected face, and if the person was identified by object class recognition the identifier matches the ID for the recognized person.
- The `region` element contains the location of the clothing. 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.
- The `regiontype` element specifies whether the region represents clothing covering the full body (`Full`), upper body (`Upper`), or lower body (`Lower`). These values correspond to those for the configuration parameter `ClothingMode`.

Clothing analysis can produce several records for each detected face. For example, if you set `ClothingMode=Full,Upper,Lower`, Media Server might produce three records. One contains the location of the clothing covering the full body, one contains the location of the clothing covering the upper body, and one contains the location of the clothing covering the lower body. Media Server does not produce a record if the region is outside the boundary of the image or clothing analysis cannot find the region.

Chapter 20: 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 with the detection of potential threats, illegal actions, or alert human operators to situations where help is required.

Introduction to Scene Analysis

Scene analysis recognizes events in video that you consider important. Typical examples of objects and events you might want to detect in CCTV footage include:

- A vehicle breaking traffic laws, for example by running a red light.
- Abandoned bags.
- Abandoned vehicles.
- Traffic congestion.
- Zone breaches and trip wire events, for example a person entering a restricted area.

Train Scene Analysis

To run scene analysis, you must create a training configuration that specifies how to detect objects and describes the events that you want to detect. A *configuration* describes all of the events that you want to detect from a single camera.

A configuration can include multiple categories. A *category* describes a single type of event for which you want to generate alarms, for example a vehicle breaching a red light or a person entering a restricted area.

Media Server includes the scene analysis training utility, a Windows application that you can use to rapidly train scene analysis. You can use the Training Utility to:

- Define regions of interest for each category in your configuration.
- Mask parts of the scene that you do not want to monitor for any category.
- 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 and generate alarms if an event occurs while the lights are red.

- Display the video being analyzed by Media Server, with an overlay that shows objects being tracked, so that you can confirm objects are 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 remain in the scene for a certain amount of time.
- Review the alarms that have been generated using your training, and classify each one as a true alarm for a specific category, or as a false alarm. The training utility can then optimize the training to minimize the number of false alarms and the number of missed alarms.

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 110.

To detect important events in video

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=SceneAnalysis
   ```

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:

   ```
   [SceneAnalysis]
   Type=SceneAnalysis
   TrainingCfgName=redlights
   ```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 21: 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 22: Image Comparison

Image comparison identifies which parts of an image have changed, by comparing the image to a reference image that is stored in the Media Server database.

Introduction

Image comparison identifies which parts of an image have changed, by comparing the image to a reference image that is stored in the Media Server database. You can run image comparison on a video file or stream - in this case Media Server compares each video frame to the reference image.

You can use image comparison to identify changes in a scene. You might want to identify new objects that have appeared, objects that have disappeared, or objects that have moved to different locations.

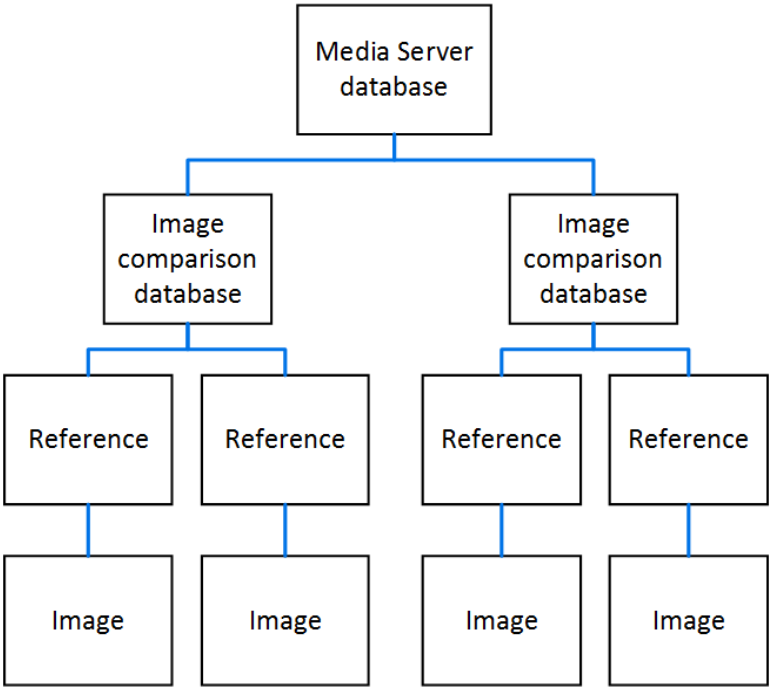
If you have deployed Media Server for security and surveillance purposes, you can capture an image of the scene from a camera and store this in the database. Then, you can compare subsequent images to the reference image and identify any changes, such a person being present in a restricted area. Having an automated process to identify changes like this can significantly reduce the amount of data that needs to be reviewed by a human operator.

The images that you send to Media Server for comparison should be similar to your reference images in terms of perspective and rotation. The images do not need to be the same size but if there are significant changes in perspective or rotation Media Server will consider the entire image to have changed.

Train Media Server to Compare Images

You must train Media Server by providing reference images as a basis for comparison.

The following diagram shows how Media Server stores the 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 Training Database, on page 42.

You can organize image comparison references into databases. These databases are for organizational purposes. When you run image comparison, you can restrict comparison to all of the references in a specific database, or to specific references in a specific database. Media Server creates a record for each comparison.

A database can contain any number of references. For each reference you must provide one training image. For information about choosing suitable training images, see Select Images for Training, below. You can also associate custom metadata with each reference.

Select Images for Training

When you train Media Server you must provide one training image for each image comparison reference.

A good training image for image comparison:

- should measure at least 400 pixels along the shortest dimension.
- contains only the part of the scene that you want to compare. If you take a reference image from a camera but only want to detect changes in part of the scene, you should crop the image before adding it to the image comparison database. When you run image comparison, you can set the `region` configuration parameter to limit analysis to the corresponding part of the ingested (source) media.
- is not blurry.

- has not been converted from another file format to JPEG. Converting images to JPEG can introduce 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 an Image Comparison Database

To create a database to contain information for image comparison, use the following procedure.

To create a database for image comparison

- Use the `CreateImageComparisonDatabase` action, with the `database` parameter:

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

  For example:

```
curl http://localhost:14000 -F action=CreateImageComparisonDatabase
                            -F database=ReferenceImages
```

Add a Reference to a Database

To add a new reference for image comparison, follow these steps.

To add a reference to a database

- Add the reference using the action `TrainImageComparisonReference`, with the following parameters:

  `database` The name of the database to add the reference to. The database must already exist.

  `identifier` (Optional) A unique identifier for the reference (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.

  `imagedata` (Set this or `imagepath`, but not both). The training image to add. The file 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 78.

  `imagepath` (Set this or `imagedata`, but not both). The path of the training image to add. The path must be absolute or relative to the Media Server executable file.

  `imagelabels` (Optional) A label to identify the image that you are adding (maximum 254 bytes). If you do not set this parameter, Media Server generates a label automatically.

  `metadata` (Optional) A comma-separated list of metadata key-value pairs to add to the reference. 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=TrainImageComparisonReference
                            -F database=ReferenceImages
                            -F imagedata=@camera39.png
                            -F metadata=Location:Cambridge
```

Alternatively, the following example provides the path of the training image rather than sending the image data:

```
curl http://localhost:14000 -F action=TrainImageComparisonReference
                            -F database=ReferenceImages
                            -F imagepath=./training/comparison/camera39.png
                            -F "metadata=Location:Cambridge"
```

Media Server adds the reference to the database and returns the identifier.

List the References in a Database

To list the image comparison references that you have added to a database, and check whether training was successful, use the following procedure.

To list the image comparison references in a database

1. (Optional) First list the databases that have been created for use with image comparison. Use the action `ListImageComparisonDatabases`:

   ```
   http://localhost:14000/action=ListImageComparisonDatabases
   ```

   Media Server returns a list of databases that you have created.

2. List the references that exist in one of the databases. Use the action `ListImageComparisonReferences`, for example:

   ```
   /action=ListImageComparisonReferences&database=ReferenceImages
                                        &imagestatus=true
                                        &metadata=true
   ```

   Media Server returns a list of references in the specified database.

   If you set the action parameter `metadata` to **`true`**, Media Server returns any metadata you have added to the references.

   If you set the action parameter `imagestatus` to **`true`**, Media Server returns the status of each training image associated with each reference.

   - The `status` element indicates the status of training:
     - `trained` indicates that training was successful.
     - `failed` indicates that Media Server could not use the image for training.

Update or Remove References and Databases

To update or remove an image comparison reference use the following actions:

- To add, remove, or update custom metadata for a reference, use the actions `AddImageComparisonMetadata`, `RemoveImageComparisonMetadata`, and `UpdateImageComparisonMetadata`.
- To retrieve the training image from an image comparison reference, use the action `GetImageComparisonReferenceImage`.
- To change the identifier of a reference you added to an image comparison database, use the action `RenameImageComparisonReference`.
- To change the label of an image that you added to a reference, use the action `RelabelImageComparisonReferenceImage`.
- To move an image comparison reference to a different database, use the action `MoveImageComparisonReference`.
- To remove a reference from a database and discard the training image and all associated metadata, use the action `RemoveImageComparisonReference`.

To update or remove change detection databases, use the following actions:

- To rename a database, use the action `RenameImageComparisonDatabase`.
- To delete an image comparison database and all of the information that it contains, use the action `RemoveImageComparisonDatabase`.

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

Compare Images

To identify differences between an image and a reference image that you added to the Media Server database, configure an image comparison analysis task by following these steps.

To compare images

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=CompareImages
   ```

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 **`ImageComparison`**.
`Input`	(Optional) The image track to process.
`Database`	(Optional) The database that contains the references to compare to. By default, Media Server uses all databases. Database names are case-sensitive.
`Identifier`	(Optional) A comma-separated list of identifiers that specifies the reference images to compare against. To use this parameter you must set the `Database` parameter and the references that you specify must exist in that database. Micro Focus recommends that you set both `Database` and `Identifier` to reduce the number of comparisons.
`Region`	(Optional) Set this parameter to compare a region of the ingested image to the reference image(s).

For example:

```
[CompareImages]
Type=ImageComparison
Database=ReferenceImages
Identifier=TransportCamera39
```

For more details about the parameters that you can use to customize image comparison, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Image Comparison Results

The following XML shows a single record produced by image comparison.

```
<output>
  <record>
    <pageNumber>1</pageNumber>
    <trackname>Compare.Result</trackname>
    <ImageComparisonResult>
      <id>60480c72-fa13-401d-92e6-e7ec87eb3e13</id>
      <identity>
        <identifier>Camera39</identifier>
        <database>ReferenceImages</database>
        <imagelabel>4132cf6ee5bcff7b03037140767ff32c</imagelabel>
        <metadata>
          <item>
            <key>Location</key>
            <value>Cambridge</value>
          </item>
        </metadata>
      </identity>
```

```
      <imageChangeScore>19.27</imageChangeScore>
      <changedRegion>
        <region>
          <left>40</left>
          <top>8</top>
          <width>40</width>
          <height>40</height>
        </region>
        <score>82.72</score>
      </changedRegion>
      <changedRegion>
        <region>
          <left>80</left>
          <top>8</top>
          <width>40</width>
          <height>40</height>
        </region>
        <score>30.14</score>
      </changedRegion>
      ...
</output>
```

The record contains the following information:

- The `identity` element describes the reference image (in your training database) that was used for comparison. Media Server creates a separate record for each reference that the ingested image was compared to.
  - The `identifier` element provides the identifier of the reference.
  - The `database` element provides the name of the database in which the reference exists.
  - The `imagelabel` element provides the label of the reference image that was used for comparison.
  - The `metadata` element provides metadata that you associated with the reference when you trained Media Server. If there is no metadata in the training database, this element is omitted.
- The `imageChangeScore` element indicates the amount of change between the images. This value is a percentage, between zero (no changes) and 100 (every region has significant changes).
- Each `changedRegion` element describes a change between the reference image (in the training database) and the source image (that was analyzed). If there are no differences, there will be no `changedRegion` elements present. Each `changedRegion` element describes a small part of the image.
  - The `region` element describes the position of the top-left corner of the region, and its width and height. The region refers to the source image, not the reference image. This is particularly important in cases where the source media is a different size to your reference image. If you set the `region` configuration parameter to analyze only part of the source media, the co-ordinates `0,0` still refer to the top-left corner of the full source media.
  - The `score` element specifies how much the region has changed, as a percentage from zero to 100. A high score indicates that there are major changes within the region and a low score

indicates minor differences. You can use the scores to locate the regions that show the greatest change.

Chapter 23: Color Clustering

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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   ...
   Engine2=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.
`ColorDictionary`	(Optional) A dictionary file that associates names with RGB color values. If you set this parameter, the task clusters colors around colors that are defined in the dictionary, and the results will include a name (such as "light blue" or "red") for each cluster. If the dictionary file is located in the `colorcluster` folder, in the static data directory, you can specify just the file name. Otherwise specify a path - either absolute or relative to the Media Server executable file.

   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 Dictionaries

You can configure the color cluster analysis task to cluster colors around colors that are defined in a dictionary. If you configure a dictionary, Media Server also returns the name of a color for each cluster in the analysis results. Media Server includes some dictionaries in the `colorcluster` folder, in the static data directory.

To list the colors that have been defined in a color dictionary

- Use the action `ListColors`. For example:

  ```
  /action=ListColors&ColorDictionary=basiccolors.dat
  ```

  where the `ColorDictionary` parameter specifies either:

  - the name of a dictionary file that is located in the `colorcluster` folder in the static data directory. The static data directory is the folder specified by the `StaticDataDirectory` parameter in the `[Paths]` section of the Media Server configuration file.
  - the path of a dictionary file. A path can be absolute or relative to the Media Server executable 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>
        <colorname>white</colorname>
        <proportion>77.12</proportion>
      </cluster>
      <cluster>
        <color>10 11 13</color>
        <colorname>black</colorname>
        <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 `colorname` element provides a name for the color. This element is present only if you have specified the path to a dictionary that associates names with color values. For information about color dictionaries, see Color Dictionaries, on the previous page.
  - The `proportion` element specifies the percentage of the image or video frame that belongs to that cluster.

In addition to the `Result` track, the color clustering task also produces a track named `ClusteredImage`. This contains the source image, containing only colors that match the center of a color cluster, and cropped to the analyzed region. If the analyzed region is not rectangular any pixels outside the region are transparent (or black if you use an image format that does not support transparency).

Chapter 24: Barcode Recognition

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
- Code-128
- Code-39
- Code-93
- Datalogic 2/5
- Data Matrix
- EAN-13 (including optional EAN-2 or EAN-5 supplement)
- EAN-8 (including optional EAN-2 or EAN-5 supplement)
- I25
- IATA 2/5
- Industrial 2/5
- Matrix 2/5
- Patch Code
- PDF417
- QR
- UPC-A (including optional EAN-2 or EAN-5 supplement)
- UPC-E (including optional EAN-2 or EAN-5 supplement)

NOTE:
Some types of barcode are a subset of other types.

- UPC-A is a subset of EAN-13. If you configure Media Server to detect EAN-13 barcodes, Media Server returns all EAN-13 barcodes including UPC-A.

- ISBN barcodes are a subset of EAN-13 that begin with 978 or 979, so you can detect ISBN barcodes by configuring Media Server to detect EAN-13 barcodes, but Media Server will also return EAN-13 barcodes than are not in the ISBN range.
- GS1-128 barcodes (sometimes known as UCC/EAN-128) are a subset of Code-128 barcodes, so you can detect these using the Code-128 type.

Read Barcodes

To read barcodes

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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. If Media Server detects a barcode with a supplement, for example EAN-13+EAN-2, the `text` element contains the digits from both parts of the barcode, separated by a hyphen.
- The `barcodeType` element contains a string which describes the type of the detected barcode. This can be any of the following values:
  - Codabar
  - Code-128
  - Code-39
  - Code-93
  - Datalogic 2/5
  - Data Matrix
  - EAN-13
  - EAN-13+EAN-2
  - EAN-13+EAN-5
  - EAN-8
  - EAN-8+EAN-2
  - EAN-8+EAN-5
  - I25
  - IATA 2/5
  - Industrial 2/5
  - Matrix 2/5
  - Patch Code
  - PDF417
  - QR
  - UPC-A
  - UPC-A+EAN-2
  - UPC-A+EAN-5
  - UPC-E
  - UPC-E+EAN-2
  - UPC-E+EAN-5
- The `region` element describes the position of the barcode in the ingested media. If Media Server detects a barcode with a supplement, for example EAN-13+EAN-2, the region includes both parts of the barcode.

Chapter 25: Generate Image Hashes

Media Server can generate an image hash from an image. This section describes how to use the image hash feature to identify duplicates in a set of images.

Introduction

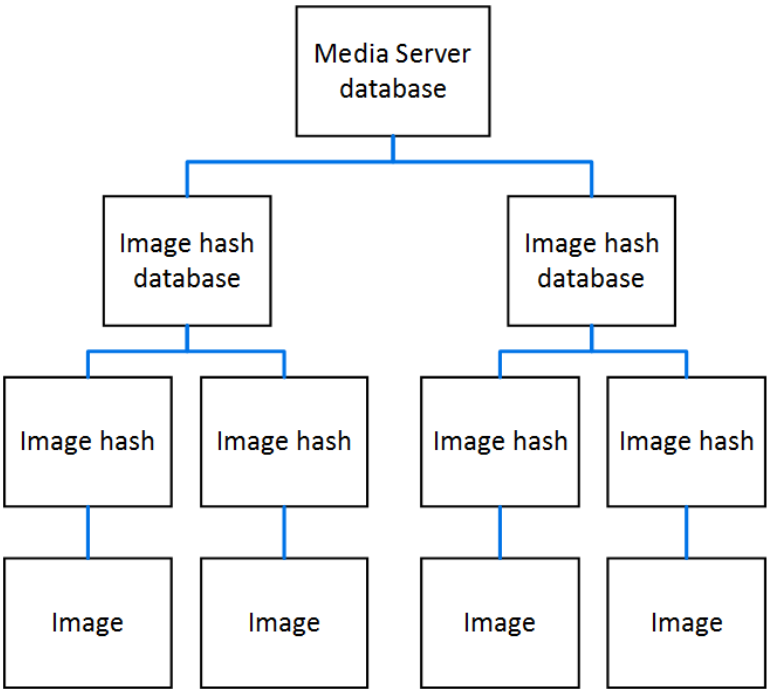
An image hash can be considered as a cursory summary of an image. You can use image hash analysis to identify exact duplicates in a set of images, because identical images produce identical hashes. Image hash analysis can tolerate small distortions, but an image that is subject to significant changes results in a different hash.

Image hash analysis is much faster than object recognition or image comparison because it considers each image as a whole and does not analyze the detail within an image. To identify which parts of an image have changed, use image comparison instead. If you need a solution that can identify matching images while tolerating distortion (including scaling, rotation, and perspective changes or skew), consider using object recognition.

Train Media Server to Identify Duplicate Images

You must train Media Server by providing reference images as a basis for comparison.

The following diagram shows how Media Server stores the 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 setting up this database, see Set up a Training Database, on page 42.

You can organize image hashes into databases. These databases are for organizational purposes. When you run the analysis task, you can restrict comparison to all of the hashes in a specific database, or to specific hashes in a specific database.

A database can contain any number of image hashes. For each image hash you must provide one training image.

Create an Image Hash Database

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

To create a database for image hashes

- Use the `CreateImageHashDatabase` action, with the `database` parameter:

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

  For example:

```
curl http://localhost:14000 -F action=CreateImageHashDatabase
                            -F database=ImageHashes
```

Add an Image Hash to a Database

To add a new image hash and train Media Server to identify duplicate images, follow these steps.

To add an image hash to a database

- Add the image hash using the action `TrainImageHash`, with the following parameters:

  `database` The name of the database to add the image hash to. The database must already exist.

  `identifier` (Optional) A unique identifier for the image hash (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.

  `imagedata` (Set this or `imagepath`, but not both). The training image to add. The file 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 78.

  `imagepath` (Set this or `imagedata`, but not both). The path of the training image to add. The path must be absolute or relative to the Media Server executable file.

  `imagelabel` (Optional) A label to identify the image that you are adding (maximum 254 bytes). If you do not set this parameter, Media Server generates a label automatically.

  For example:

```
curl http://localhost:14000 -F action=TrainImageHash
                            -F database=ImageHashes
                            -F imagedata=@image1.png
```

  Alternatively, provide the path of the training image:

```
curl http://localhost:14000 -F action=TrainImageHash
                            -F database=ImageHashes
                            -F imagepath=./training/image1.png
```

  Media Server adds the image hash to the database and returns the identifier.

List the Image Hashes in a Database

To list the image hashes that you have added to a database, use the following procedure.

To list the image hashes in a database

1. (Optional) First list the databases that have been created. Use the action `ListImageHashDatabases`:

   ```
   http://localhost:14000/action=ListImageHashDatabases
   ```

   Media Server returns a list of databases that you have created.

2. List the image hashes that exist in one of the databases. Use the action `ListImageHashes`, for

example:

```
/action=ListImageHashes&database=ImageHashes
                        &imagestatus=true
```

Media Server returns a list of image hashes in the specified database.

Update or Remove Image Hashes and Databases

To update or remove an image hash use the following actions:

- To retrieve the training image from an image hash, use the action `GetImageHashImage`.
- To change the identifier of an existing image hash, use the action `RenameImageHash`.
- To change the label of the image associated with an image hash, use the action `RelabelImageHashImage`.
- To move an image hash to a different database, use the action `MoveImageHash`.
- To remove an image hash from a database and discard the training image, use the action `RemoveImageHash`.

To update or remove image hash databases, use the following actions:

- To rename a database, use the action `RenameImageHashDatabase`.
- To delete a database and all of the information that it contains, use the action `RemoveImageHashDatabase`.

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

Identify Duplicate Images

To identify duplicate images

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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 that contains the images to analyze.
`Database`	(Optional) The image hash database to use to identify duplicate images. If you do not set this parameter, Media Server uses all image hash databases.
`MatchThreshold`	(Optional) The minimum confidence score required for a match.

For example:

```
[ImageHash]
Type=ImageHash
Database=MyImageHashes
MatchThreshold=60
```

For a complete list of parameters you can use to customize this task, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example Configuration

The following example configuration ingests an image file, analyzes the image to see whether it is a match for any of the image hashes in a database named `ImageHashes`, and then outputs the results to the ACI response.

```
[Session]
Engine0=Ingest
Engine1=ImageHash
Engine2=OutputACI

[Ingest]
Type=Image

[ImageHash]
Type=ImageHash
Database=ImageHashes
MatchThreshold=50

[OutputACI]
Type=Response
Input=ImageHash.Result
```

If you add this configuration to the configurations directory as `ImageHash.cfg` you could run a `process` action as follows (where the `source` parameter specifies the image to analyze):

```
http://localhost:14000/action=process&configname=ImageHash
                                    &source=./image_to_analyze.png
```

Image Hash Results

The following XML shows a single record produced by image hash analysis:

```
<output>
  <record>
    <pageNumber>1</pageNumber>
    <startTime iso8601="2018-01-05T11:02:49.038357Z">1515150169038357</startTime>
    <trackname>ImageHash.Result</trackname>
    <ImageHashResult>
      <id>5536224b-4fdf-4fee-bb7a-fe47eee9f010</id>
      <identity>
        <identifier>MyImage</identifier>
        <database>ImageHashes</database>
        <confidence>100</confidence>
      </identity>
    </ImageHashResult>
  </record>
</output>
```

Media Server creates a separate record for each image hash that an ingested image is compared to.

The `identity` element of the record describes the image hash (in your training database) that was used for comparison. This element contains the following information:

- The `identifier` element provides the identifier of the image hash.
- The `database` element provides the name of the database.
- The `confidence` element provides the confidence score for the match, from 0 to 100 where 100 indicates maximum confidence. You can configure Media Server to discard results with a low confidence score by setting the parameter `MatchThreshold` (see Identify Duplicate Images, on page 240).

Chapter 26: Audio Categorization

Media Server can categorize audio. Audio categorization segments and classifies audio into predefined categories such as "speech", "music", "noise", and "silence".

You can use audio categorization to inspect an audio file and decide whether to perform further processing. For example, when audio categorization reports that a file contains mostly speech, you might decide to run language identification and speech-to-text.

If you are processing recordings of telephone calls, you can use audio categorization to identify DTMF tones.

Categorize Audio

To run audio categorization

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=AudioCategorize
   ```

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 **`AudioCategorize`**.
`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.
`SampleFrequency`	(Optional) The sample frequency of the audio to send to the audio service for analysis, in samples per second (Hz).

   For example:

   ```
   [AudioCategorize]
   Type=AudioCategorize
   SampleFrequency=16000
   ```

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

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Audio Categorization Results

The following XML shows a single record produced by audio categorization.

```
<record>
  <timestamp>
    <startTime iso8601="1970-01-01T00:00:01Z">1000000</startTime>
    <duration iso8601="PT00H00M00.500000S">500000</duration>
    <peakTime iso8601="1970-01-01T00:00:01Z">1000000</peakTime>
    <endTime iso8601="1970-01-01T00:00:01.500000Z">1500000</endTime>
  </timestamp>
  <trackname>AudioCategorize.Result</trackname>
  <AudioCategorizeResult>
    <id>e8d84838-bdf2-4b9b-9a92-e7e42b249103</id>
    <category>Music</category>
    <confidence>80</confidence>
  </AudioCategorizeResult>
</record>
```

The record contains the following information:

- The `id` element contains the identifier for the audio segment.
- The `category` element shows how the audio segment was classified. The categories are pre-defined and this value can be:
  - `DialTone`
  - `DTMF-*`, `DTMF-0`, `DTMF-1`, `DTMF-2`, and so on. These values indicate that the audio contains a DTMF tone. For example, `DTMF-2` indicates the tone for the "2" button.
  - `Music`
  - `Noise`
  - `Silence`
  - `Speech`

  > **NOTE:**
  > Dial tone and DTMF tone detection are enabled only when you process audio with a sample rate of 8KHz.

- The `confidence` element provides the confidence score for the classification, from 0 to 100, where 100 indicates the greatest confidence.

Chapter 27: Language Identification

Language identification identifies the language of speech.

Identify the Language of Speech

To identify the language of speech

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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.
`Languages`	(Optional) The list of languages to consider when running language identification. If you know which languages are likely to be present in the media, Micro Focus recommends setting this parameter because restricting the possible languages can increase accuracy and improve performance. For a list of supported languages with language codes, see Speech Analysis Supported Languages, on page 383.
`Mode`	The type of language identification task to run. Set this parameter to one of the following options: • **`Boundary`** - Language identification seeks to determine boundaries in the audio where the language changes, and returns results for the time between boundaries. • **`Segmented`** - The audio is segmented into fixed-size segments and Media Server does not consider previous segments when running analysis. You can use this mode to determine the language if there are multiple languages present in the audio,

but this mode does not identify the exact boundary points at which the language changes.

- **`Cumulative`** - 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. You might use this mode if you are processing a video file and expect the audio to contain only one language or you want to identify the primary language that is spoken.

  > **NOTE:**
  > Cumulative mode is not suitable for analyzing continuous streams.

`SegmentDuration` (Optional, default `15s`) The amount of audio to analyze as a single segment.

For example:

```
[SpeechLanguageId]
Type=LanguageID
Languages=ENUK,DEDE
Mode=Cumulative
SegmentDuration=30s
```

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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 28: Speaker Identification

Speaker identification identifies the people who speak in audio or video.

Train Speaker Identification

Speaker identification divides audio into different speakers. Media Server can identify the gender of each speaker without training, but to recognize individual speakers you must train Media Server by providing audio samples for each person.

Micro Focus recommends that you provide at least five minutes of speech for each speaker. An audio sample must not contain speech from any other speakers. Ideally, you should use high-quality audio samples that contain only the speaker's voice and no background noise. However, you should include samples from a range of environments (indoors, outdoors, noisy, and so on) that match what you expect to process. The audio sample can contain any vocabulary - the speaker does not need to say any specific phrase.

If you want to process audio that includes unknown speakers (people who you have not trained and do not exist in the database) there are some additional training requirements:

- You must provide audio samples that represent unknown speakers (any speakers you have not trained; the audio samples you provide for unknown speakers do not need to match the unknown speakers in the audio you are going to process). Micro Focus recommends that you provide at least 60 minutes of audio containing unknown speakers. This audio must not contain any of the speakers that you have trained.
- You must provide additional audio samples for each of the speakers you have trained, to be used as *development* rather than training samples. The development samples must be different to the training samples. Micro Focus recommends that you provide at least five minutes of speech for each speaker.

These additional audio samples are used to generate the thresholds that Media Server uses to distinguish between a match to a known speaker and an unknown speaker.

The speakers that you train are organized into databases. When you run speaker identification you provide the name of the database to use and Media Server attempts to recognize speakers against the speakers in that database. For example, you could create a database named "news" for processing news broadcasts and train various speakers (newsreaders, politicians, and so on) who you expect to appear.

A television news broadcast is an example that contains unknown speakers, because you cannot expect to predict who will speak or provide audio samples for every person. So in this case you would need to provide development audio samples for each speaker you train, and add audio samples to the database that represent unknown speakers.

Create a Speaker Database

To create a new speaker database, use the following procedure.

To create a speaker database

- Use the `CreateSpeakerDatabase` action with the following parameters.

`database`	The name of the new database (maximum 254 bytes).
`samplefrequency`	The sample frequency of the audio that this speaker database can be used with. Specify either 8000 (8kHz telephony) or 16000 (16kHz broadband). The default is 16000.

For example:

```
curl http://localhost:14000 -F action=CreateSpeakerDatabase
                            -F database=news
                            -F samplefrequency=16000
```

Add Speakers to a Database

To add a speaker to a speaker database, follow these steps.

To add a speaker

1. Create a new speaker using the `NewSpeaker` action. Set the following parameters:

`database`	The name of the database to add the speaker to. The database must already exist.
`identifier`	(Optional) A unique identifier for the speaker (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.

For example:

```
curl http://localhost:14000 -F action=NewSpeaker
                            -F database=news
                            -F identifier=newsreader
```

Media Server adds the speaker and returns the identifier.

2. Add audio samples for the speaker using the `AddSpeakerAudio` action. Set the following parameters:

`database`	The name of the database that contains the speaker.
`identifier`	The identifier for the speaker, as returned by the `NewSpeaker` action.

`audiodata`	(Set this or `audiopath`, but not both). The audio data 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 78.
`audiopath`	(Set this or `audiodata`, but not both). A comma-separated list of paths to the audio files. The paths must be absolute or relative to the Media Server executable file.
`audiolabels`	(Optional) A comma-separated list of labels to identify the audio samples that you are adding (maximum 254 bytes for each label). Every audio sample added to the same speaker must have a unique label, so the number of labels must match the number of samples provided using either `audiodata` or `audiopath`. If you do not set this parameter, Media Server generates labels automatically.

For example:

```
curl http://localhost:14000 -F action=AddSpeakerAudio
                            -F database=news
                            -F identifier=newsreader
                            -F audiodata=@sample1.wav,sample2.wav
                            -F audiolabels=sample1,sample2
```

3. Train Media Server to recognize the speaker by running the `BuildSpeaker` action. Set the following parameters:

`database`	The name of the database that contains the speaker.
`identifier`	The identifier for the speaker, as returned by the `NewSpeaker` action.

For example:

```
curl http://localhost:14000 -F action=BuildSpeaker
                            -F database=news
                            -F identifier=newsreader
```

Generate Speaker Thresholds

This section describes how to generate the speaker thresholds that are used to distinguish between known and unknown speakers.

You only need to complete the steps in this section if you want to process audio that contains unknown speakers (people who you have not trained and do not exist in the database). For more information about training speaker identification, see Train Speaker Identification, on page 247.

To generate speaker thresholds

1. For each of the speakers you have trained, add additional audio samples for generating speaker thresholds. To add these audio samples, use the `AddSpeakerAudio` action but set the parameter

`training=false`. For example:

```
curl http://localhost:14000 -F action=AddSpeakerAudio
                            -F database=news
                            -F identifier=newsreader
                            -F audiodata=@sample3.wav,sample4.wav
                            -F audiolabels=sample3,sample4
                            -F training=FALSE
```

2. Add audio samples that represent unknown speakers, using the action `AddUnknownSpeakerAudio`. Set the following parameters:

`database`	The name of the database to add the audio samples to.
`audiodata`	(Set this or `audiopath`, but not both) The audio data 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 78.
`audiopath`	(Set this or `audiodata`, but not both) A comma-separated list of paths to the audio files to add. The paths must be absolute, or relative to the Media Server executable file.
`audiolabels`	(Optional) A comma-separated list of labels to identify the audio samples that you are adding (maximum 254 bytes for each label). Every audio sample representing unknown speakers must have a unique label, so the number of labels must match the number of samples provided using either `audiodata` or `audiopath`. If you do not set this parameter, Media Server generates labels automatically.

For example:

```
curl http://localhost:14000 -F action=AddUnknownSpeakerAudio
                            -F database=news
                            -F audiodata=@UnknownSpeakers.wav
```

3. Calculate speaker thresholds, by running the action `EstimateAllSpeakerThresholds`. Set the `database` parameter to specify the name of the database. For example:

```
curl http://localhost:14000 -F action=EstimateAllSpeakerThresholds
                            -F database=news
```

Optimize Speaker Thresholds

When you process audio that includes unknown speakers, Media Server uses speaker thresholds to distinguish between a match to a known speaker and an unknown speaker. Micro Focus recommends that you set speaker thresholds using the action `EstimateAllSpeakerThresholds`, as described in Generate Speaker Thresholds, on the previous page

If necessary, you can optimize the thresholds that Media Server sets:

- The `EstimateAllSpeakerThresholds` and `EstimateSpeakerThreshold` actions have a parameter named `bias`. Increasing the value of this parameter increases the thresholds that are generated,

which reduces the probability of false positives but could increase the number of missed results (where a known speaker is classified as unknown). Decreasing the value of the `bias` parameter has the opposite effect (fewer missed results but potentially more false positives).

- You can use the action `SetSpeakerThreshold` to manually specify a threshold for a speaker. If you run speaker identification and Media Server incorrectly identifies a speaker you can compare the confidence score in the speaker identification output to the speaker threshold and change the threshold appropriately. To obtain the current threshold for each speaker, use the action `ListSpeakers`.

List the Speakers in a Database

To list the speakers that you have added to a database, use the following procedure.

To list the speakers in a database

1. (Optional) First list the speaker databases that have been created. For example:

   ```
   http://localhost:14000/action=ListSpeakerDatabases
   ```

   Media Server returns a list of speaker databases.

2. List the speakers that exist in one of the databases. For example:

   ```
   http://localhost:14000/action=ListSpeakers&database=news
                                            &audiostatus=true
   ```

   Media Server returns a list of speakers in the specified database. For more information about this action and the response, refer to the *Media Server Reference*.

Identify Speakers

To identify speakers

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=RecognizeSpeakers
   ```

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.
`Database`	The name of the database to use to recognize speakers. If you do not set this parameter Media Server cannot identify speakers, but can divide the audio into different speakers and detect the gender of each speaker.
`ClosedSet`	A Boolean value (default false) that specifies whether the audio contains only known speakers (who are in the database specified by the `Database` parameter). If the audio that you are processing contains only known speakers, set this parameter to **`true`**.
`SampleFrequency`	(Optional, default 16000) The sample frequency of the audio to send to the audio service for analysis, in samples per second (Hz).

For example:

```
[RecognizeSpeakers]
Type=SpeakerID
Database=news
ClosedSet=false
SampleFrequency=16000
```

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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 29: Speech-to-Text

Speech-to-text converts words spoken in audio and video into text.

Introduction

Media Server can perform Speech-to-Text, which extracts speech from the audio and converts it into text. When the audio or video source contains narration or dialogue, running Speech-to-Text 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.
- categorize video clips.
- cluster together video clips that contain related concepts.

Custom Language Models

Speech-to-text is supported out-of-the-box and does not require training, but you can often improve accuracy by creating a custom language model. A *language model* is just one part of the training for a language. It describes the vocabulary and contains information about how sentences are composed from individual words. This section explains when you might want to use a custom language model and how to build one.

Using a custom language model can improve accuracy when:

- The audio contains specialized vocabulary. Any language model has a finite vocabulary and your audio might contain words that are not in the standard vocabulary. For example, a recording of a lecture to medical professionals might include specialized medical terminology, and a recording of a telephone call to a customer support center might include product names.
- You have access to many transcripts that are representative of typical conversations, such as call center conversations.

Building a language model requires a lot of text - millions or billions of words. The language models supplied with Media Server are trained with billions of words across a wide range of topics. This is a significant training burden, but you can build a small, focused, language model and use it to supplement one of the standard models.

Standard language model	Custom language model
• Available out of the box. • Trained on billions of words. • Covers a wide range of topics. • Might not cover specialist vocabulary.	• You need to create it yourself. • Generally trained on a smaller number of words. The custom language model is combined with a standard language model, to increase the coverage. • Focuses on particular topics, to increase accuracy. • Can cover specialist vocabulary, such as technical terms or product names.

Select Text for Training

To create an effective custom language model, you need sample text that strongly resembles the speech you want to process. For example, if you use speech-to-text for news monitoring, you could train a language model using recent news articles gathered from the web.

To process a recording of a lecture to medical professionals, you could source text from:

- Transcripts of similar lectures.
- Articles written by the speaker who delivered the lecture.
- Slides used in delivering the lecture.
- Any other document related to the topic or event.
- A web article that discusses the particular topic.

Other useful sources of text include:

- Transcripts of call conversations.
- Literature that describes a product or company.
- Company websites that contain natural language descriptions (rather than images or advertisements).

The amount of text required to build a custom language model can vary from a few thousand words to several hundred thousand words, depending on the topic. Generally, using more text to build the custom language model increases accuracy. However, the gains in accuracy reduce after you exceed a certain number of words (approximately 100,000 words for a typical topic such as technical support).

The training text does not need to cover all the vocabulary in the audio. The custom language model is combined with a standard language model, so that the coverage is the sum of the two models. Therefore, building a custom language model using small quantities of text still provides benefits.

Prepare Text for Training

You might have to clean up your sample text before you can use it for training.

To prepare sample text

- Remove anything that does occur in spoken language, such as HTML tags and tables.
- Ensure that sentence breaks (periods) are present in appropriate places.
- Ensure that there are no duplicated sections in the text.
- Ensure the text is encoded in UTF-8.

Text must also be normalized. Media Server automatically normalizes the text for most languages. You only need to normalize text yourself when automatic normalization is not supported for your language, or you set `normalize=false` when you train the custom language model.

To normalize text

- Change digits to words. For example, replace "2" with "two". Replace "37" with "thirty seven" (not "three seven"). Replace the year "1997" with "nineteen ninety seven".
- Write individually pronounced character sequences as spaced characters; for example, “the word rules is spelled R U L E S”.
- Write pronounced punctuation as it sounds; for example, “sales at Micro Focus dot com".
- For all sentence breaks, replace periods (.) with <s>. Other punctuation must be removed.

Train a Custom Language Model

To create a new custom language model, use the action `TrainCustomSpeechLanguageModel`.

To train a custom language model

- Train a custom language model using the `TrainCustomSpeechLanguageModel` action. Set the following parameters:

`identifier`	(Optional) A unique identifier for the custom language model (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.
`languagepack`	The language to base the custom language model on. You cannot train a custom language model with one language and use it with another, so this parameter and the `LanguagePack` parameter in your speech-to-text task must have the same value.
`textdata`	(Set this or `textpath`, but not both). The text to use for training. Text 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 78.
`textpath`	(Set this or `textdata`, but not both). A comma-separated list of paths to the files that contain the text to use for training. The paths must be absolute, or relative to the Media Server executable file.

For example:

```
curl http://localhost:14000 -F action=TrainCustomSpeechLanguageModel
                            -F languagepack=ENUK
                            -F textdata=@sample1.txt,sample2.txt
```

Alternatively, the following example provides the paths of the text files, rather than sending the data:

```
curl http://localhost:14000 -F action=TrainCustomSpeechLanguageModel
                            -F languagepack=ENUK
                            -F textpath=data/sample1.txt,data/sample2.txt
```

You can list the custom language models that you have trained using the action `ListCustomSpeechLanguageModels`. For more information about the actions that you can use for training custom language models, refer to the *Media Server Reference*.

Assess Language Models

Before running speech-to-text you can assess whether a language pack, optionally combined with a custom language model, is suitable for processing your audio. You can check:

- whether the words that you want to recognize are included in the vocabulary. Media Server cannot recognize words unless they are in the vocabulary. If your audio has many words that are not in the vocabulary (for example product names), you can increase accuracy by training a custom language model.
- the estimated branching factor for the words in the speech (the *perplexity*). A lower value is generally better. Call center conversations typically follow a set pattern and therefore have a lower perplexity than broadcast audio.

To check whether words are present in the vocabulary

- Obtain sample text that closely resembles the speech you want to process. Then, submit the text to the action `AssessSpeechLanguageModel`. Media Server returns statistics and information about unknown words.
- Compile a list of words that you know you need to recognize and use the `QuerySpeechLanguageModel` action to check whether the words are present in the vocabulary.

To measure perplexity for a language model

- Obtain sample text that closely resembles the speech that you want to process. Then, submit the text to the action `AssessSpeechLanguageModel`. Media Server returns a perplexity value. Perplexity values around or below 100 are acceptable for processing call center conversations. Perplexity values around or below 250 are acceptable for television news/broadcast audio. A lower perplexity value is generally better. If the `AssessSpeechLangaugeModel` action returns a perplexity value that is much higher, consider training a custom language model.

For more information about the `AssessSpeechLanguageModel` and `QuerySpeechLanguageModel` actions, refer to the *Media Server Reference*.

Transcribe Speech

To run speech-to-text

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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.
`LanguagePack`	The language pack to use. For a list of available language packs, see Speech Analysis Supported Languages, on page 383.
`CustomLanguageModel`	(Optional) A comma-separated list of custom language models to use. For each custom language model, specify the identifier and interpolation weight, separated by a colon.
`SpeedBias`	Specifies whether to prioritize accuracy or speed. If you are processing a live stream, you must set this parameter to `Live`. Otherwise, set this parameter to an integer from `1` to `6`, where `1` prioritizes accuracy and `6` prioritizes speed.
`FilterMusic`	(Optional, default `false`) Specifies whether to ignore speech-to-text results for audio segments that are identified as music or noise. To filter these results from the output, set this parameter to **`true`**.
`SampleFrequency`	(Optional, default `16000`) The sample frequency of the audio to send to the audio service for analysis, in samples per second (Hz). Language packs are dependent on the audio sample rate, and accept audio at either 8000Hz or 16000Hz.

   For example:

   ```
   [TranscribeSpeech]
   Type=SpeechToText
   LanguagePack=ENUK
   CustomLanguageModel=MedicalTerms:0.1
   ```

```
SpeedBias=2
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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Pre-Load Language Resources

Media Server automatically loads any resources that are needed to run speech-to-text, but processing cannot start until the resources have finished loading and, due to the amount of data, this might be 15 or 20 seconds after the `process` action is received.

You can load language resources before you send a `process` action, so that the resources required by the session configuration are ready and processing can begin immediately. This is particularly beneficial when you process live streams, to avoid missing the start of a broadcast.

Any language resources that you load remain in memory until you unload them, so pre-loading resources can sometimes help to increase throughput when you process many audio or video files with the same configuration.

> **NOTE:**
> A language resource is a combination of a language pack and zero or more custom language models with specific interpolation weights. For example, the following would all be loaded as separate language resources:
>
> - The `ENUK` language pack (British English, for audio with a 16kHz sample rate) without a custom language model.
> - The `ENUK` language pack (British English, for audio with a 16kHz sample rate) combined with a custom language model named `MedicalTerms` with an interpolation weight of `0.1`.
> - The `ENUK` language pack (British English, for audio with a 16kHz sample rate) combined with a custom language model named `MedicalTerms` with an interpolation weight of `0.2`.

Micro Focus recommends that you pre-load the language resources that you expect to use frequently, and allow Media Server to load other language resources on demand.

To load language resources, use the action `LoadSpeechLanguageResource`. Any language resources you load with this action remain in memory until you unload them or until Media Server is stopped.

To load language resources automatically when Media Server starts, set the `SpeechLanguageResources` parameter in the `[PersistentData]` section of the Media Server configuration file.

You can list the language resources that you have loaded by running the action `ListSpeechLanguageResources`, and unload them by running the action `UnloadSpeechLanguageResources`.

For more information about these actions and configuration parameters, refer to the *Media Server Reference*.

Chapter 30: Audio Matching

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.

Train Audio Matching

You must train Media Server by providing the audio clips that you want to recognize. Each clip you provide is added to a database. Add all of the clips that you want to use in the same task to the same database.

Create a Database to Contain Audio Clips

To create a database to contain known audio clips, use the following procedure.

To create a database to contain audio clips

- Use the action `CreateAudioMatchDatabase`:

`database`	The name of the new database (maximum 254 bytes).
`samplefrequency`	(Optional) The sample frequency of the audio that this audio clip database can be used with. Specify either `8000` (8kHz telephony) or `16000` (16kHz broadband). The default is `16000`.

For example:

```
curl http://localhost:14000 -F action=CreateAudioMatchDatabase
                            -F database=Music
```

Add a Clip to the Database

After creating a database, train Media Server to recognize each audio clip by running the action `TrainAudioMatchClip`.

To add an audio clip to the database

- Add a clip using the action `TrainAudioMatchClip`. Set the following parameters:

`database`	The name of the database to add the clip to. The database must already exist.
`identifier`	(Optional) A unique identifier for the clip (maximum 254 bytes). If you do not set this parameter, Media Server generates an identifier automatically.
`audiodata`	(Set this or `audiopath`, but not both). The audio data. 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 78.
`audiopath`	(Set this or `audiodata`, but not both). The path of the audio clip. The path must be absolute or relative to the Media Server executable file.

  For example:

  ```
  curl http://localhost:14000 -F action=TrainAudioMatchClip
                              -F database=Music
                              -F audiodata=@clip1.wav
  ```

  Alternatively, the following example provides the path of the audio clip rather than sending the audio data:

  ```
  curl http://localhost:14000 -F action=TrainAudioMatchClip
                              -F database=Music
                              -F audiopath=clip1.wav
  ```

  Media Server adds the clip to the database and returns the identifier.

List the Clips in a Database

To list the audio clips that you have added to a database, use the following procedure.

To list the clips in a database

1. (Optional) First list the databases that have been created for audio matching. Use the action `ListAudioMatchDatabases`:

   ```
   http://localhost:14000/action=ListAudioMatchDatabases
   ```

   Media Server returns a list of databases.

2. List the clips that exist in one of the databases. Use the action `ListAudioMatchClips`, for example:

   ```
   http://localhost:14000/action=ListAudioMatchClips&database=music
   ```

   Media Server returns a list of clips in the specified database.

Manage Audio Clips and Databases

To remove an audio clip from a database, use the action `RemoveAudioMatchClip`.

To update or remove databases, use the following actions:

- To rename an audio match database, use the action `RenameAudioMatchDatabase`.
- To delete an audio match database and all of the information that it contains, use the action `RemoveAudioMatchDatabase`.

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

Recognize Audio Clips

To recognize audio clips

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new analysis task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=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.
`Database`	The name of the audio match database to use for recognizing audio clips.
`SampleFrequency`	(Optional) The sample frequency of the audio to send to the audio service for analysis, in samples per second (Hz).

   For example:

   ```
   [AudioMatch]
   Type=AudioMatch
   Database=Music
   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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 31: Transcript Alignment

Transcript alignment aligns a transcript with the speech in a media file.

Introduction

Transcript alignment takes a transcript of the speech in a media file and, by processing the speech, assigns timestamps to all the words in the transcript. This is useful because it allows an application to provide search results from the transcript and open the media file at the correct position. You can also synchronize manually created subtitle text with the speech in a video file.

The transcript does not need to be an exact match for the speech. Media Server can tolerate small numbers of errors in the transcript, and some background noise and music in the audio. However, transcript alignment is intended to be used when you already have a transcript that contains the words that are spoken (such the script for a television broadcast). If you do not have a transcript you can create one manually; otherwise, you might prefer to run speech-to-text instead. There is no advantage in running transcript alignment on a transcript that was produced by speech-to-text (unless you find and correct any errors in the speech-to-text output).

The transcript must be normalized, but Media Server automatically normalizes transcripts in many languages. You only need to normalize the transcript yourself when automatic normalization is not supported for the language you are using. If you need to normalize the transcript yourself, use the same procedure as described for custom language models (see Prepare Text for Training, on page 254). Then, run the `AlignAudioTranscript` action again but send the normalized text and set `normalize=false`.

Run Transcript Alignment

To run transcript alignment you must have installed a speech-to-text language pack that matches the language of the speech. You must also enable the speech-to-text module. For information about how to install speech-to-text language packs, see Install Speech-to-Text Language Packs, on page 40. For information about how to enable and disable modules, see Specify Modules to Enable, on page 70.

To run transcript alignment, follow these steps.

To run transcript alignment

1. Prepare a transcript of the speech in your media file. The transcript should be a plain text file.
2. Send the transcript and the media file to the `AlignAudioTranscript` action. For example:

```
curl http://localhost:14000 -F action=AlignAudioTranscript
                            -F audiodata=@audio.wav
                            -F textdata=@transcript.txt
```

```
-F languagepack=ENUS
-F samplefrequency=16000
```

Media Server returns a token. You can use the token with the `QueueInfo` action to retrieve the results. For more information about the `AlignAudioTranscript` action, refer to the *Media Server Reference*.

Chapter 32: 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 327.

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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` 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:

   ```
   [Session]
   Engine0=Ingest
   Engine1=SpeechToText
   Engine2=TextSegmentation
   Engine3=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 Speech-to-Text, on page 253.
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. Micro Focus 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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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=AV

[AV]
Type=Video

[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`.

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 33: 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

To encode video to one or more MPEG files on disk, follow these steps.

To encode video to a file

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `EngineN` parameter. You can give the task any name, for example:

```
[Session]
Engine0=Ingest
Engine1=MyEncodingTask
```

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

`Type`	The type of encoding engine to use. 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 ensure Media Server uses the same aspect ratio as the source video, set either the width or height and set the other dimension to zero.
`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. Specify the path as an absolute path or relative to the Media Server executable file.
`UrlBase`	The base URL that will be used to access the encoded files. This is used when Media Server generates proxy information. 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.

   For example:

```
[MyEncodingTask]
Type=mpeg
VideoSize=0x720

OutputPath=./mp4/%currentTime.year%/%currentTime.month%/%currentTime.day%/%curr
entTime.timestamp%_%segment.sequence%.mp4

UrlBase=http://www.mysite.com/video/clips/mp4/%currentTime.year%/%currentTime.m
onth%/%currentTime.day%/
```

   For more information about the configuration parameters and macros that you can use, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Encode Video to a UDP Stream

To generate a live UDP stream of the content that Media Server is ingesting, use the following procedure.

To encode video to a stream

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the

task any name, for example:

```
[Session]
Engine0=Ingest
Engine1=EncodeStream
```

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

`Type`	The type of encoding engine to use. 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 ensure Media Server uses the same aspect ratio as the source video, set either the width or height and set the other dimension to zero.
`OutputURL`	The URL to stream to. For example, `udp://239.255.1.123:4321`.
`Format`	The container format. For example, `mpegts` for MPEG transport stream.

For example:

```
[EncodeStream]
Type=mpeg
VideoSize=0x720
OutputUrl=udp://239.255.1.123:4321
Format=mpegts
```

For more information about the configuration parameters that you can use to configure this task, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 34: 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 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 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, Micro Focus 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&Stream=NewsChannel&MaxFiles=10&MaxFileSize=100
  ```

  where the `Stream` 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&Stream=NewsChannel
  ```

  where the `Stream` 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=RollingBuffer
   ```

3. Create a new configuration section to contain the task settings, and 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 276).

`AudioProfile` The audio encoding profile to use.

`VideoProfile` The video encoding profile to use.

For example:

```
[RollingBuffer]
Type=RollingBuffer
Stream=NewsChannel
AudioProfile=mpeg4audio
VideoProfile=mpeg4video_h264_sd
```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Write Video to an Evidential Rolling Buffer

To write ingested video to an evidential rolling buffer, follow these steps.

To write video to an evidential rolling buffer

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=EvidentialRollingBuffer
   ```

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

   `Type` The type of engine to use. Set this parameter to **`EvidentialRollingBuffer`**.

   `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 276).

   For example:

   ```
   [EvidentialRollingBuffer]
   Type=EvidentialRollingBuffer
   Stream=Camera3
   ```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 as a time duration.

  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=24hours
  ```

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 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.
`Duration`	(Optional) The length of time after the start time that the playlist covers.
`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=5minutes
```

  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.
`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=5minutes
                                        &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 35: 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, Micro Focus 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=KeyframeAnalysis
   Engine2=ImageEncoder
   ```

3. Create a new configuration section to contain the task settings, and 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. Specify the path as an absolute path or relative to the Media Server executable file.
`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.

For example:

```
[ImageEncoder]
Type=ImageEncoder
ImageInput=KeyframeAnalysis.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*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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.

- Event Stream Processing

Chapter 36: 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	Combine tracks from OCR,

	output track. The output track contains all of 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.
AndAny	Compares two tracks and produces a track containing records from the first track for which there is at least one event in the second track within a specified time interval (before or after the first track's event).	
AndThenAny	Compares two tracks and produces a track containing records from the first track which are followed within a specified time interval by	

	at least one event in the second track.	
Combine	Combines records from two or more tracks. 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 other input track(s). In comparison, the And task creates one output record for each pair of related records. • Every record from the first input track always appears in the output, even if there are no related records in the other input track(s).	Combine information about detected faces that appear at the same time so that all of the faces can be blurred by a transformation task.

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 303.

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`, `AndThenAny`, 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.

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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine5=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 303.

5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example

The following configuration produces a new track called `Weather.Output`. This track only contains records that include the word "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 class recognition	Result, ResultWithSource	The recognizer, 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 303.

To deduplicate a track

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine5=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 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 303.

   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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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).

```
[DeduplicateOCR]
Type=deduplicate
```

```
Input=myocr.data
MinTimeInterval=1000ms
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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine5=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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example

The following configuration combines output tracks from a Barcode task and an OCR task:

```
[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 297.

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 303.

To identify time-linked events in two tracks

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine5=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 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 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. 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 303.

`LuaScript`	The path and file name of a Lua script to run.

5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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.

```
[Session]
...
Engine5=BreakingNews
Engine6=Syria
Engine7=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=2000ms
```

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 295.

To identify time-linked events in two tracks

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine5=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 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 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 303.

For more information about these parameters and the values they accept, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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.

```
[RedLightBreach]
Type=andthen
Input0=sceneanalysis.ResultWithSource
Input1=numberplate.ResultWithSource
MaxTimeInterval=5000ms
```

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 the next page).

To identify isolated events

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine5=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 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 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. 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 303.

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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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.

```
[Session]
...
Engine5=FilterAudio
Engine6=LogoWithoutCompanyName

[FilterAudio]
Type=filter
...

[LogoWithoutCompanyName]
Type=andnot
Input0=RecognizeCompanyLogo.Result
Input1=FilterAudio.output
MaxTimeInterval=5000ms
```

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 298).

To identify isolated events

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ```

```
...
Engine5=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 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 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 303.

   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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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.

```
[Barrier]
Type=andnotthen
Input0=sceneanalysis.Result
Input1=numberplate.Result
MaxTimeInterval=30000ms
```

Identify and Combine Time-Related Events

The *Combine* ESP engine combines records from two or more tracks. The engine produces an output track that contains exactly one record for every record in the first (`Input0`) input track. Each of the

output records contains all related records from the other input track(s). The records from the first input track are output even if there are no related events in the other track(s).

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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine5=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.
`InputN`	The other input tracks (`Input1`, `Input2`, and so on). You must specify at least `Input1`.
`MaxTimeInterval`	The maximum difference in time between a record in the first track to a record in another, for the records to be considered related. If you are processing images or documents this parameter is ignored.
`MinTimeInterval`	(Optional) The minimum difference in time between a record in the first track to a record in another, for the records to be considered related. The default value is the negative of the `MaxTimeInterval` value, meaning that an event can occur before the event in the `Input0` track. 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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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.

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=Combine
Engine3=Blur
Engine4=ToDisk

[Ingest]
Type=image

[FaceDetect]
Type=FaceDetect
FaceDirection=any
Orientation=any

[Combine]
Type=combine
Input0=Image_1
Input1=FaceDetect.Result

[Blur]
Type=Blur
Input=Combine.Output

[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	Engine response if the function returns true

	record parameters	
And	2	The record pair is included in the engine output track.
AndThen	2	The record pair is included in the engine output track.
AndAny	2	The record in the first track is included in the output. (Records from the second track are never included in the output).
AndThenAny	2	The record in the first track is included in the output. (Records from the second track are never included in the output).
AndNot	2	The record in the first track is discarded. (Records from the second track are never included in the output).
AndNotThen	2	The record in the first track is discarded. (Records from the second track are never included in the output).
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.

- Crop Images
- Blur Regions of Images
- Draw Regions
- Create Overlays
- Rotate Images
- Resize Images
- Change the Format of Images

Chapter 37: 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 to send to Media Server with the process action, or open an existing configuration that you want to modify.
2. In the [Session] section, add a new task by setting the Engine*N* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine2=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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 38: 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine3=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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example Configuration

The following is an example configuration that blurs faces in a source image and encodes the resulting image to disk.

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=Combine
Engine3=Blur
Engine4=ToDisk
Engine5=ACI

[Ingest]
Type=Image

[FaceDetect]
Type=FaceDetect

[Combine]
// Combine FaceDetect results in case there are multiple faces in the image
Type=Combine
Input0=Default_Image
Input1=FaceDetect.Result

[Blur]
Type=Blur
Input=Combine.Output

[ToDisk]
Type=ImageEncoder
ImageInput=Blur.Output
OutputPath=./output/%token%/blurred-faces-%source.filename%

[ACI]
Type=Response
Input=FaceDetect.Result
```

Chapter 39: Draw Regions

Many analysis tasks identify regions in images and video frames. In some cases you might want to draw these regions on the images or video before encoding. For example, you can produce a video where detected faces are surrounded by an ellipse or number plates are surrounded by a polygon.

Introduction

The **Draw** transformation task draws regions, that were identified during analysis, on images or video frames.

By default, Media Server draws the regions described in the following table:

Face detection	An ellipse that describes the position of a face. The ellipse is described by the `ellipse` element in the records created by face detection.
Number plate recognition	A polygon that describes the position of the numberplate. The polygon is described by the `plateregion` element in the records created by number plate recognition.
Other tracks where the records contain regions	If the record contains a polygon, Media Server draws the polygon. Otherwise Media Server draws the bounding rectangle for the region.

To draw outlines around multiple types of region, for example both detected faces and recognized logos, you can combine multiple tracks using a combine ESP task. The output from the combine ESP task, and therefore the input for the drawing task, is a track where records contain other records. In this case the drawing task iterates over the nested records.

You can configure the line color and line thickness that Media Server uses for drawing.

You can also customize the drawing task by writing a Lua script that defines the regions to draw. For example, if you run face detection and demographic analysis, you can write a Lua script to outline male and female faces in different colors.

The `Draw` task creates an output track named *`taskName`*`.Output`, where *`taskName`* is the name of the task. The images in this track are permanently modified (the regions that you draw on the images cannot be removed).

Draw Regions

The **Draw** transformation task draws regions on images or video frames.

To draw regions on images and video frames

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine2=Draw
   ```

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 **`Draw`**.
`Input`	The name of the track that contains the images to draw on, with region data. The track must supply records that contain both an image and at least one region.
`Color`	(Optional) The line color to use when drawing.
`Thickness`	(Optional) The line thickness to use when drawing. Specify the thickness in pixels.

   For example:

   ```
   [Draw]
   Type=Draw
   Input=CombineESP.Output
   Color=Red
   Thickness=1
   ```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example configuration to ingest video and draw regions on encoded images

The following example configuration ingests a video file or stream and encodes one image for each detected face. Each image will show a red ellipse around the detected face.

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=Draw
Engine3=Images

[Ingest]
Type=Video

[FaceDetect]
Type=FaceDetect
NumParallel=6
```

```
SizeUnit=percent
MinSize=10

[Draw]
Type=Draw
Input=FaceDetect.ResultWithSource
Color=Red
Thickness=2

[Images]
Type=ImageEncoder
ImageInput=Draw.Output
OutputPath=./output/%session.token%/%record.starttime.iso8601%.png
```

Example configuration to ingest video and draw regions on encoded video

The following example configuration ingests a video file or stream and encodes a video that shows detected faces by surrounding them with a red ellipse.

In this example, an ESP task combines the `Image_1` track and the `FaceDetect.Data` track. This is necessary because the tracks produced by face detection only contain frames with detected faces, but to encode the video correctly we need to encode every frame.

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=Combine
Engine3=Draw
Engine4=MPEG

[Ingest]
Type=Video

[FaceDetect]
Type=FaceDetect
NumParallel=6
SizeUnit=percent
MinSize=10

[Combine]
Type=Combine
Input0=Image_1
Input1=FaceDetect.Data
MaxTimeInterval=50ms
[Draw]
Type=Draw
Input=Combine.Output
Color=Red
Thickness=2

[MPEG]
```

```
Type=mpeg
VideoProfile=mpeg4video_h264_720p
ImageInput=Draw.Output
OutputPath=./output/%session.token%/%segment.startTime.iso8601%.ts
```

Configure Drawing with a Lua Script

The **Draw** transformation task draws regions on images or video frames. This section demonstrates how to customize the regions that are drawn by using a Lua script.

The Lua script that you write must define a function with the name `draw`. The function is passed each record from the input track. For example, the following function draws an ellipse on each record from the `ResultWithSource` track produced by face detection:

```
function draw(record)
   drawEllipse(record.FaceResultAndImage.face.ellipse, 3, rgb(255,255,255))
end
```

For information about the Lua functions that you can use to draw regions on images and video frames, refer to the *Media Server Reference*.

To configure drawing with a Lua script

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine3=Draw
   ```

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 **Draw**.
`Input`	The name of the track that contains the images to draw on, with region data. The track must supply records that contain both an image and at least one region.
`LuaScript`	The path of a Lua script to run to draw on the source images or video frames.

   For example:

   ```
   [Draw]
   Type=Draw
   Input=Demographics.ResultWithSource
   LuaScript=lua/drawDemographics.lua
   ```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example configuration to draw regions on encoded images using a Lua script

An example configuration file, `configurations/examples/Other/Draw_Faces.cfg`, is included in your Media Server installation.

The following example runs face detection and demographics analysis on a video file or stream. It encodes images of the detected faces, using a Lua script to outline male faces in orange and female faces in purple.

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=Demographics
Engine3=Draw
Engine4=Images

[Ingest]
Type=Video

[FaceDetect]
Type=FaceDetect
NumParallel=6
SizeUnit=percent
MinSize=10

[Demographics]
Type=Demographics
Input=FaceDetect.ResultWithSource
NumParallel=2

[Draw]
Type=Draw
Input=Demographics.ResultWithSource
LuaScript=drawDemographics.lua

[Images]
Type=ImageEncoder
ImageInput=Draw.Output
OutputPath=./output/%session.token%/%record.starttime.iso8601%.png
```

The Lua script `drawDemographics.lua` is included below:

```
function draw(record)
  local result = record.DemographicsResultAndImage

  if ('Male' == result.gender) then
    -- draw orange ellipse
    drawEllipse(result.face.ellipse, 5, rgb(255, 128, 0))
```

```
  elseif ('Female' == result.gender) then
    -- draw purple ellipse
    drawEllipse(result.face.ellipse, 5, rgb(64, 0, 128))

  else
    -- draw grey ellipse
    drawEllipse(result.face.ellipse, 5, rgb(128, 128, 128))

  end
end
```

Example Lua script to process combined/nested records

To draw outlines around multiple types of region, for example both detected faces and recognized logos, you can combine multiple tracks using a combine ESP task. The output from the combine ESP task, and the input for the drawing task, is therefore a track where records contain other records. In this case, your Lua script must process the nested records. The following script demonstrates how to do this:

```
function dispatchResult(record)
	-- handle the single records here
end

function uncombine(record)
	if (record.CombineOperationData) then
		local i = 1
		while (record.CombineOperationData.combinedRecords[i]) do
			uncombine(record.CombineOperationData.combinedRecords[i])
			i = i + 1
		end
		uncombine(record.CombineOperationData.record0)
	else
		dispatchResult(record)
	end
end

function draw(record)
	uncombine(record)
end
```

Chapter 40: Create Overlays

Many analysis tasks identify regions in video. In some cases you might want to encode a video with an overlay that identifies these regions. For example, you can produce a video where detected faces are surrounded by an ellipse or number plates are surrounded by a polygon.

Creating an overlay produces similar results to drawing on the video with the draw transform engine. However, the overlay is written to the subtitle track so the video frames are not modified and the overlay can be turned on and off when you play the video in the MMAP Media Player.

Overlays are supported only for video. If you want to draw on images, use the draw transform engine.

Create an Overlay

The **Overlay** transformation task creates an overlay for a video. The task draws regions (such as rectangles, polygons, and ellipses) that are present in the input records.

To create an overlay for a video

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine2=Overlay
   ```

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 **`Overlay`**.
`Input`	The name of the track that contains the metadata to use to draw the overlay. This track does not need to include the video frames.
`Color`	(Optional) The line color to use when drawing.
`Thickness`	(Optional) The line thickness to use when drawing. Specify the thickness in pixels.

   For example:

   ```
   [Overlay]
   Type=Overlay
   Input=FaceDetection.Result
   ```

```
Color=Red
Thickness=1
```

4. When you define your encoding task, use the parameter `OverlayInput` to specify the name of the output track from the overlay transformation task. In this example the track is named `Overlay.Output`.
5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example configuration to ingest video and create an overlay

The following example configuration ingests a video file or stream and encodes a video with an overlay that shows detected faces.

```
[Ingest]
IngestEngine=AV

[AV]
Type=Video

[Analysis]
AnalysisEngine0=FaceDetect

[FaceDetect]
Type=FaceDetect
NumParallel=6
SizeUnit=percent
MinSize=10

[Transform]
TransformEngine0=Overlay

[Overlay]
Type=Overlay
Input=FaceDetect.Result
Color=Orange
Thickness=1

[Encoding]
EncodingEngine0=RollingBuffer

[RollingBuffer]
Type=RollingBuffer
ImageInput=Image_1
AudioInput=None
OverlayInput=Overlay.Output
Stream=stream1
```

Create an Overlay with a Lua Script

The `Overlay` transformation task creates an overlay for a video. This section demonstrates how to draw a custom overlay using a Lua script.

The Lua script that you write must define a function with the name `draw`. The function is passed each record from the input track. For example, the following function draws an ellipse using each record from the `Result` track produced by face detection:

```
function draw(record)
   drawEllipse(record.FaceResult.face.ellipse, 3, rgb(255,255,255))
end
```

For information about the Lua functions that you can use to draw an overlay, refer to the *Media Server Reference*.

To create an overlay with a Lua script

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine3=Overlay
   ```

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 **`Overlay`**.
`Input`	The name of the track that contains the metadata to use to draw the overlay. This track does not need to include the video frames.
`LuaScript`	The path of a Lua script to run to draw the overlay.

   For example:

   ```
   [Overlay]
   Type=Overlay
   Input=FaceDemographics.Result
   LuaScript=./configurations/overlayDemographics.lua
   ```

4. When you define your encoding task, use the parameter `OverlayInput` to specify the name of the output track from the overlay transformation task. In this example the track is named `Overlay.Output`.
5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example configuration to create an overlay using a Lua script

An example configuration file, `configurations/examples/Other/Overlay_Faces.cfg`, is included in your Media Server installation. This example ingests a video file or stream, performs face detection and demographics analysis, and encodes video with an overlay. The overlay is drawn by a Lua script that uses metadata from the demographics task to outline male faces in orange and female faces in purple.

Chapter 41: Rotate Images

You might want to rotate ingested images before analysis, or rotate images that have been analyzed so that they are in the correct orientation before being encoded. This section explains how to rotate images.

Introduction

The `rotate` transformation engine duplicates an existing track that contains images, and rotates the images (the input track is not modified). The engine also updates coordinates that are contained in records, so that the positions remain correct after the images have been rotated. The new track is named *`taskName`*`.Output`, where *`taskName`* is the name of the transformation task.

You might want to rotate images when:

- You want to process an image that is upside down and need to rotate the image before it is analyzed. Though you can configure some analysis engines to work regardless of orientation, you might obtain better performance by rotating the ingested image and restricting analysis to a single orientation.
- You run analysis tasks that detect faces or text in an image, and you want to make sure that copies of the image are encoded in the correct orientation.

Rotate Images

To configure Media Server to rotate images, follow these steps.

To rotate images

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine3=Rotation
   ```

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 **`Rotate`**.

`Input`	The name of the image track to process.
`Angle`	(Set this or `LuaScript`) The rotation to apply to the input image, in degrees, clockwise.
`LuaScript`	(Set this or `Angle`) The path to a Lua script that returns the rotation to apply to the input image, in degrees, clockwise. The Lua script must define a function named `getAngle`, which must return the angle. Media Server can rotate images in 90-degree increments, so all other angles are rounded.

For example:

```
[Rotation]
Type=Rotate
Input=FaceDetect.ResultWithSource
LuaScript=Rotate.Lua
```

A suitable Lua script is included below:

```
function getAngle(record)
  if (record.OCRResultAndImage) then
     return -record.OCRResultAndImage.angle
  elseif (record.FaceResultAndImage) then
     return -record.FaceResultAndImage.face.ellipse.angle
  elseif (record.FaceRecognitionResultAndImage) then
     return -record.FaceRecognitionResultAndImage.face.ellipse.angle
  elseif (record.DemographicsResultAndImage) then
     return -record.DemographicsResultAndImage.face.ellipse.angle
  elseif (record.FaceStateResultAndImage) then
     return -record.FaceStateResultAndImage.face.ellipse.angle
  end

  return 0
end
```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 42: 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 to send to Media Server with the process action, or open an existing configuration that you want to modify.
2. In the [Session] section, add a new task by setting the Engine*N* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine2=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	(Set this or ScaleFactor) 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.

`ScaleFactor` (Set this or `ImageSize`) The scale factor to use to calculate the size of the output image. Specify a number or a fraction. For example, to resize images to one third of the original dimensions, set `ScaleFactor=1/3`.

For example:

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

or, using a scale factor:

```
[ScaleKeyFrames]
Type=Scale
Input=Keyframe.ResultWithSource
ScaleFactor=1/2
```

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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 43: 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).

To change the format of images

1. Create a new configuration to send to Media Server with the process action, or open an existing configuration that you want to modify.
2. In the [Session] section, add a new task by setting the Engine*N* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   ...
   Engine2=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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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 44: Introduction

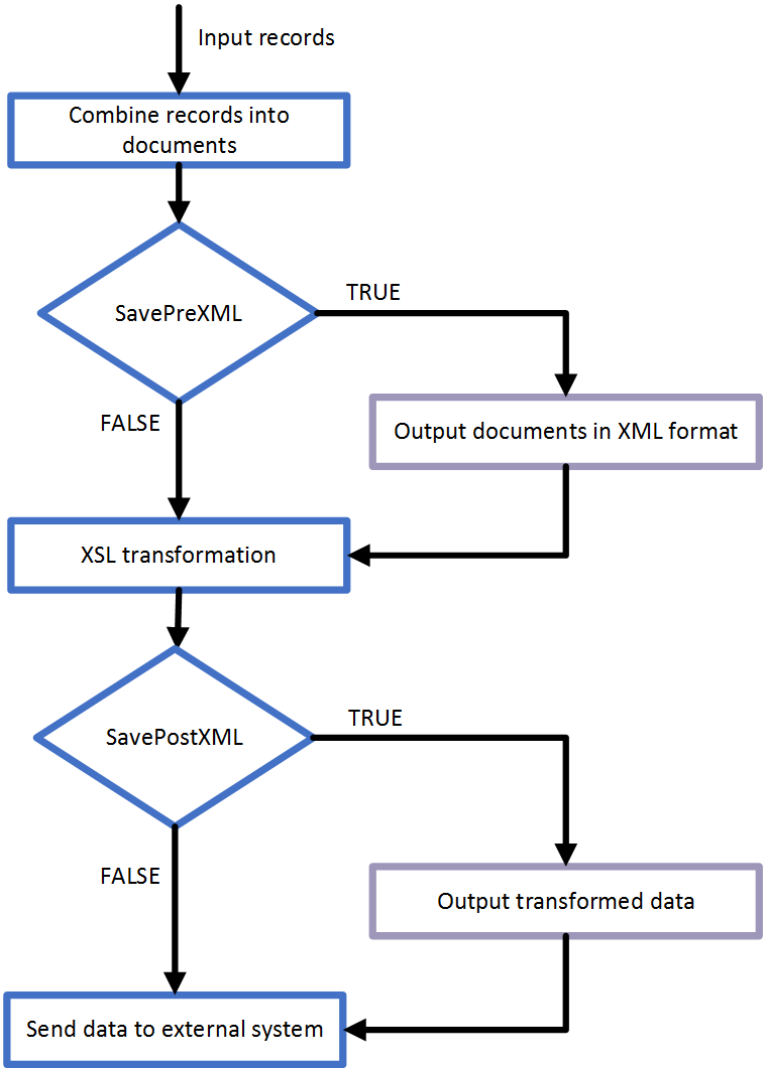
Media Server can output data in many formats, including XML and documents that you can send to CFS or index into IDOL Server.

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 272

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 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, Micro Focus 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, you might want to combine the data from many records to produce documents that represent video segments or clips.

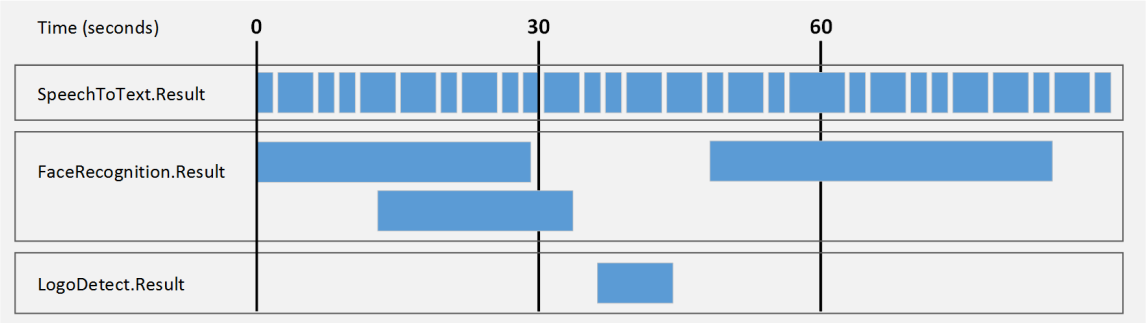
For example, the speech-to-text analysis engine produces a record for each word in the audio. If you are indexing data into a SQL database, you could insert a separate row into a database table for each word. If you are indexing data into IDOL Server, you might prefer to combine records so that each 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.

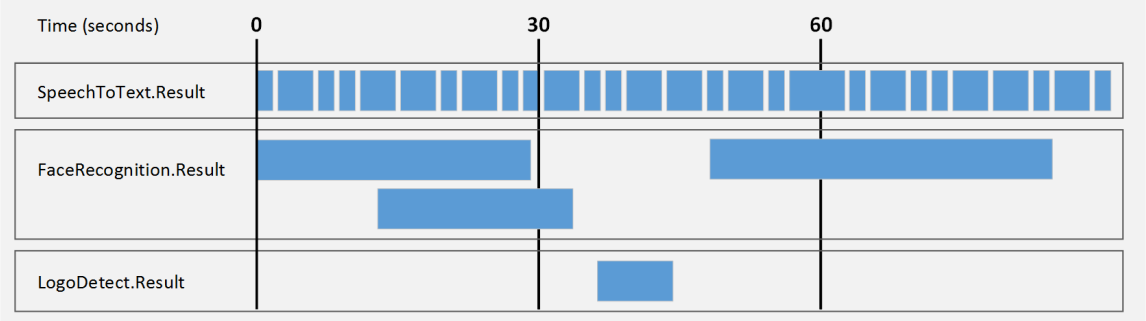
Time Mode

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.

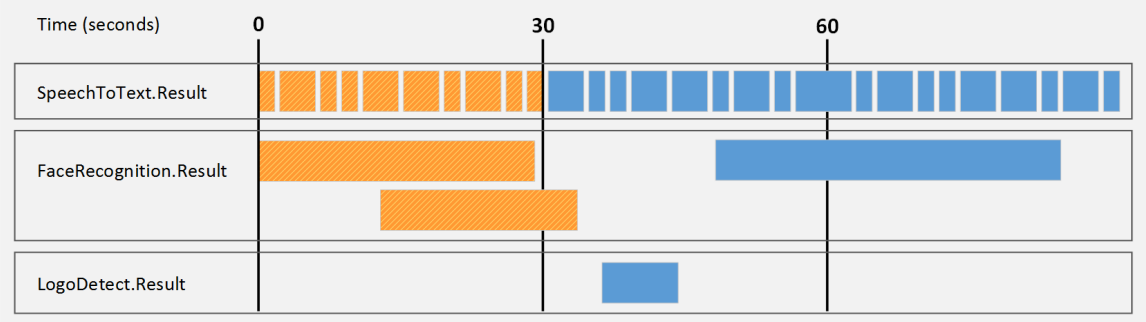
> **TIP:**
> Time-based output does not mean that Media Server outputs data at regular intervals. Media Server cannot output data for a video segment until all records in that segment have finished. As a result, Media Server might produce several documents, which represent consecutive time periods, at 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.

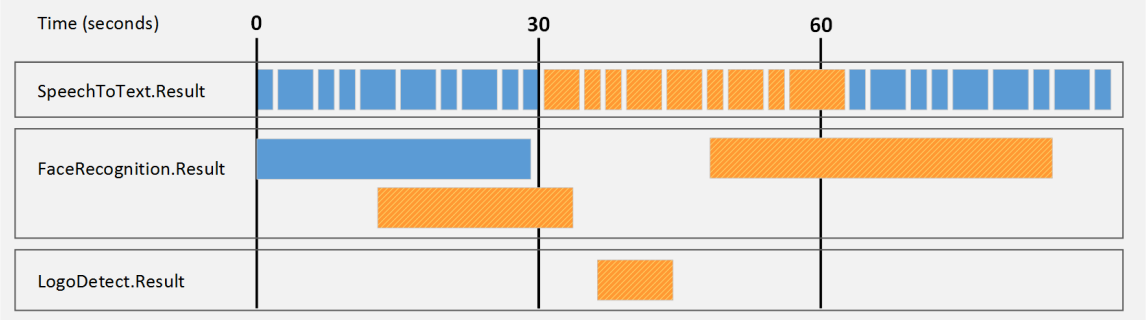
The following diagrams demonstrate how Media Server constructs documents in time mode. Consider the following records:



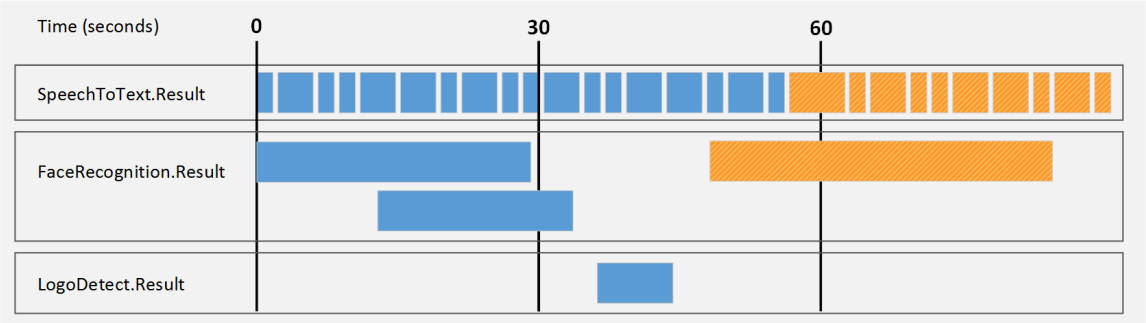
The first document contains the records related to the first interval (in this case, 30 seconds) of video content. Media Server does not output data at the 30-second point because a face recognition record has not ended (the face is still present in the video). Media Server can output data about the 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.

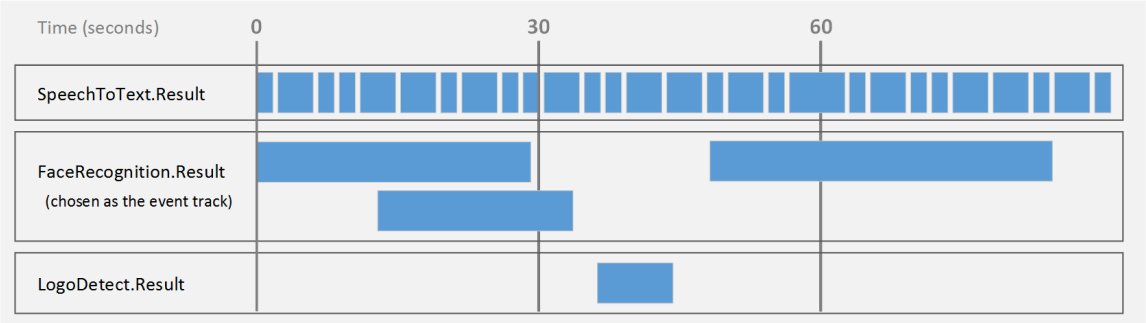
In event mode, Media Server creates a document for each record in an *event track*. The document that is generated contains the event from the event track. Other records are included if they overlap with the event in the event track or if they occurred after the end of the previous event. Each document 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.

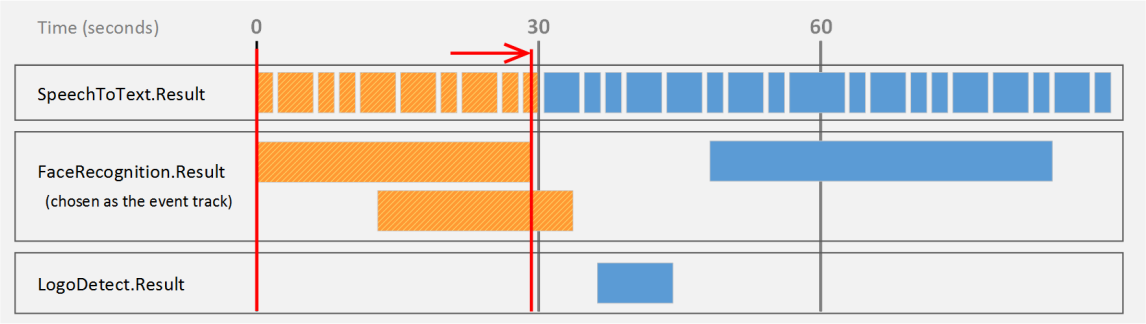
In event mode, all of the records in the selected tracks are output to at least one document. Compare this with Bounded Event Mode, on page 335, in which some records can be omitted.

> **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.

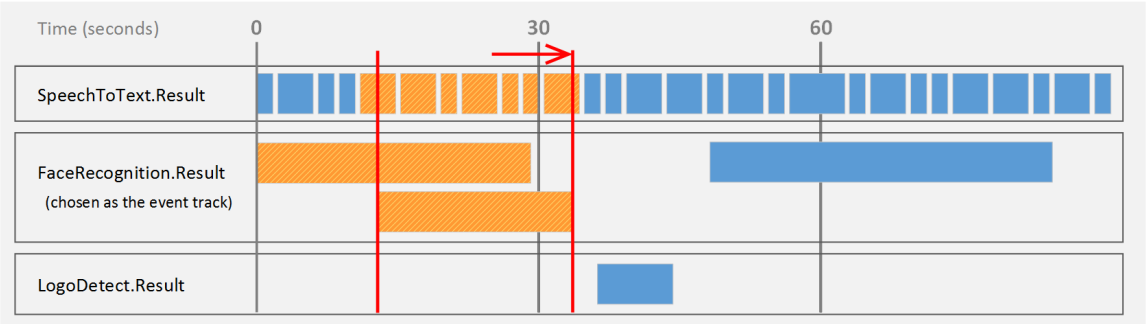
The following diagrams demonstrate how Media Server constructs documents in **event** mode. The `FaceRecognition.Result` track has been chosen as the event track. Consider the following records:



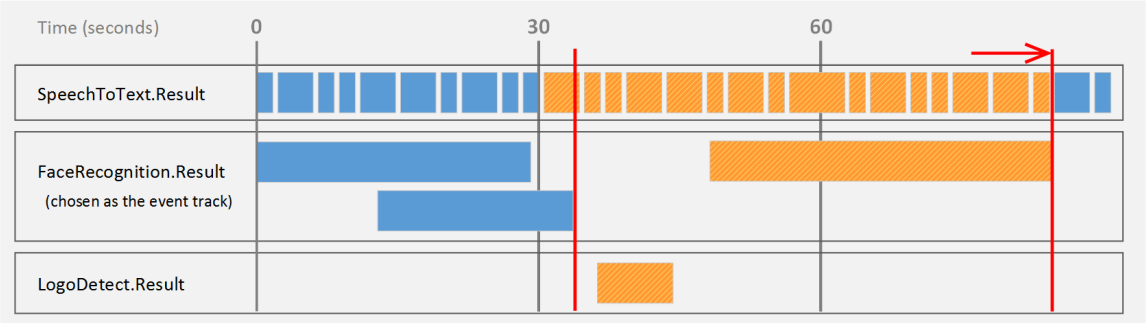
The first document contains the first record in the event track (the first event). Other records are included if they overlap with this event or if they occurred since the end of the previous event. In this case the previous event is the beginning of the video:



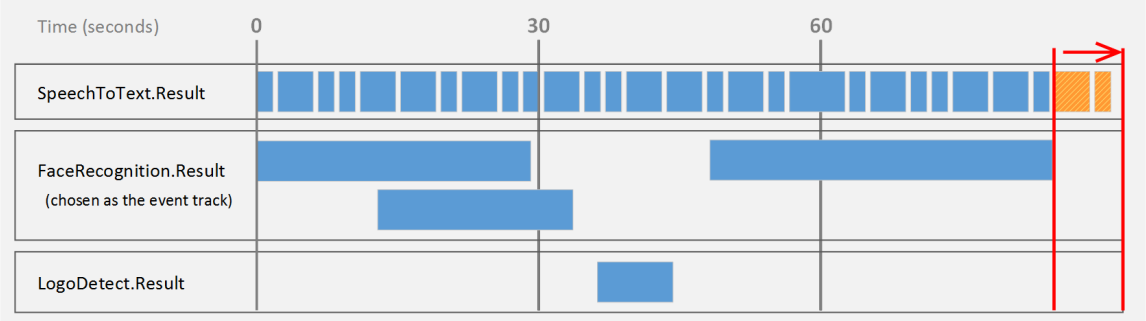
The second document contains the second record in the event track (the second event). Other records are included if they overlap with this event or if they occurred since the end of the previous event:



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

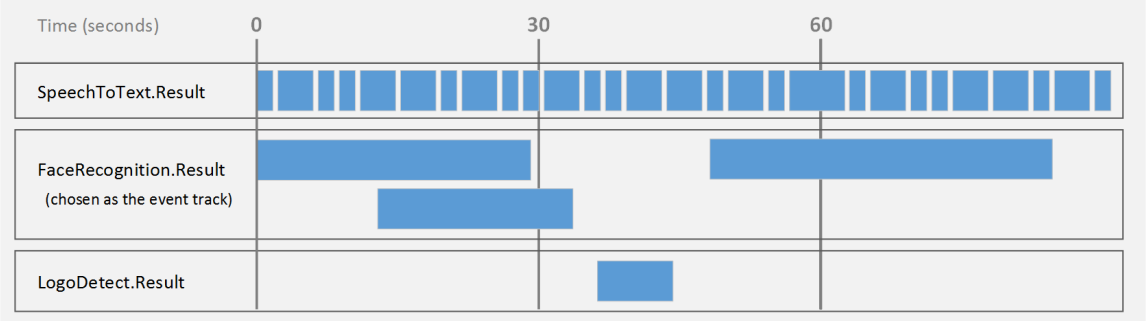
Bounded event mode, like event mode, helps to create documents that contain information about a single topic. Media Server creates a document for each record in an *event track*. The document that is generated contains the event from the event track. However, unlike event mode, other records are included 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.

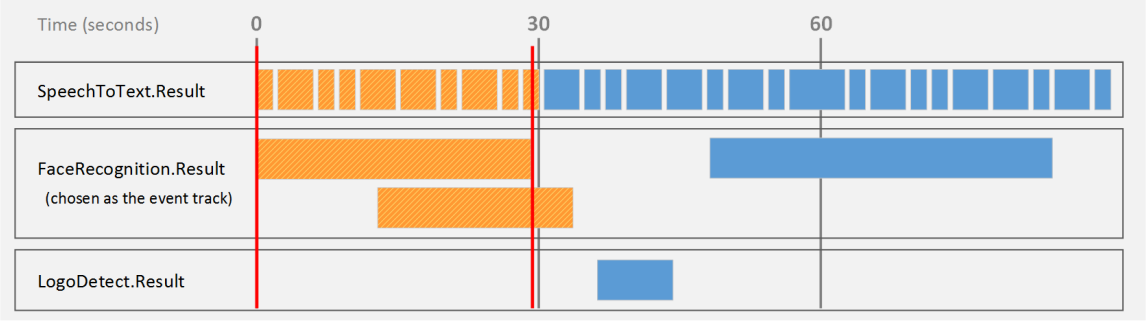
The event track can be any track that you choose, though often it will be a track that you create using event stream processing. You could use speaker identification results as your event track, so that a document is created for each speaker detected in the audio.

Each document might represent a different amount of video content.

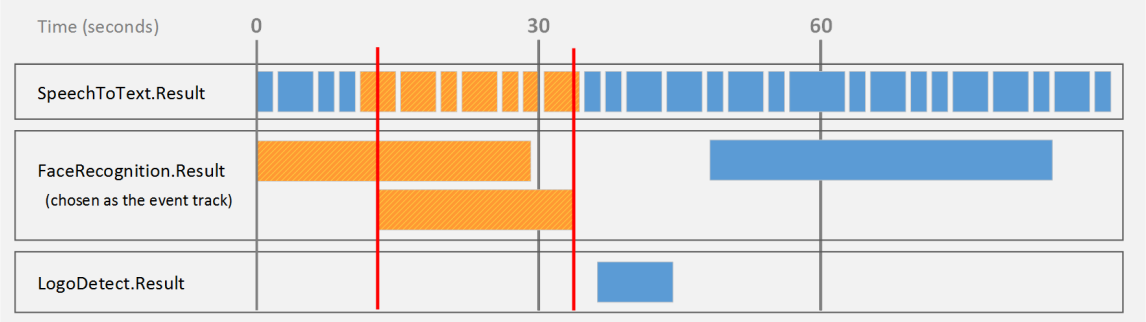
The following diagrams demonstrate how Media Server constructs documents in **bounded event** mode. The `FaceRecognition.Result` track has been chosen as the event track. Consider the following records:



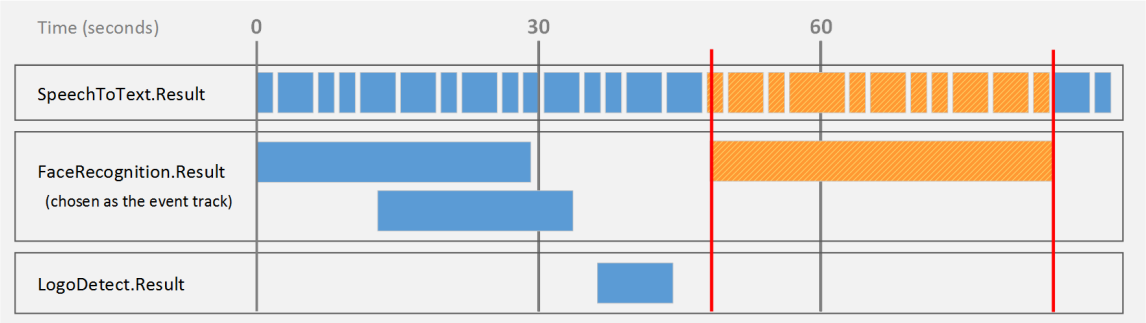
The first document contains the first record that occurs in the event track, and all of the records that occur at the same time:



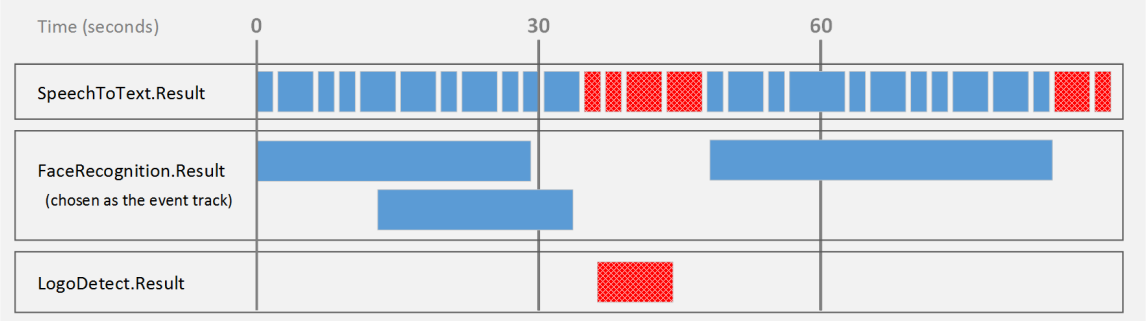
The second document contains the second record that occurs in the event track, and those records that overlap with it. Notice that if records in the event track overlap, the output document can contain more than one record from the event track:



The next document contains the next record from the event track, and the records that overlap with it:



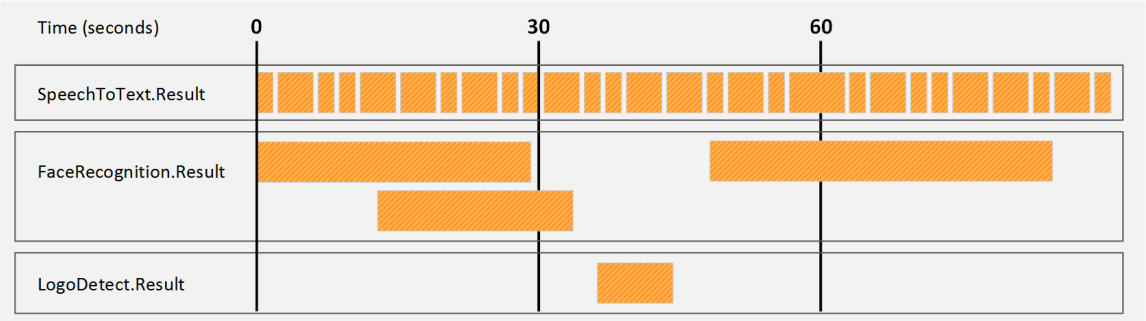
In bounded event mode, the records shown here are not output to any document, because they do not overlap with a record in the event track:



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 45: 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:**
> Micro Focus 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Response
   ```

3. Create a new configuration section to contain the task settings, 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. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example

The following is an example configuration to output data to the `process` action response.

```
[Response]
Type=response
Input=OCR.Result,Keyframe.Result
```

Chapter 46: 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=XmlWriter
   ```

3. Create a new configuration section to contain the task settings, and 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. Specify the path as an absolute path or relative to the Media Server executable file.
`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*.

4. Configure how to combine records into XML files. For information about how you can combine

records, see Choose How to Output Data, on page 330.

- 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`.

5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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=toHTML.xsl
Mode=Time
OutputInterval=30s
```

Chapter 47: Connector Framework Server

This section describes how to configure Media Server to send IDOL documents to Connector Framework Server (CFS).

Introduction

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 to convert the words that are 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.

> **NOTE:**
> The CFS output engine is intended for cases where media is ingested from an external source but you want to send documents to CFS for further processing before they are indexed into IDOL.
>
> Do not use this output engine if the source media originated from CFS and you need to return the analysis results. To return analysis results for media files that originated from CFS, configure a response output task instead. (CFS sends `process` actions to Media Server and expects the analysis results to be returned in the ACI response).

For more information about using CFS, refer to the *Connector Framework Server Administration Guide*.

Send Documents to Connector Framework Server

To send documents to CFS, use the following procedure.

To send documents to CFS

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=CFS
   ```
3. Create a new configuration section to contain the task settings, and 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 (`configurations/xsl/toCFS.xsl`) 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*.

4. Configure how to combine records into documents. For information about how you can combine records, see Choose How to Output Data, on page 330.
   - 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`.
5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example

The following is an example configuration to send documents to CFS.

```
[CFS]
Type=CFS
CfsHost=localhost
CfsPort=7000
XSLTemplate=toCFS.xsl
Mode=time
OutputInterval=20s
```

Chapter 48: 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=IDOL
   ```
3. Create a new configuration section to contain the task settings, and 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. This might be useful if you want to customize your XSL template. 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. Specify the path as an absolute path or relative to the Media Server executable file.

4. Configure how to combine records into documents. For information about how you can combine records, see Choose How to Output Data, on page 330.
   - 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`.
5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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=30s
XslTemplate=toIDX.xsl
SavePreXML=true
SavePostXML=true
XMLOutputPath=./Output/%segment.type%_%segment.sequence%_%segment.timestamp%.xml
```

Chapter 49: 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=VerticaOutput
   ```
3. Create a new configuration section to contain the task settings, and 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 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=toCSV.xsl
OutputInterval=120s
```

For more information about the parameters that you can set to configure a Vertica output task, refer to the *Media Server Reference*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Chapter 50: 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 `configurations/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, Micro Focus 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=MyDatabase
   ```

3. Create a new configuration section to contain the task settings, and 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.

4. 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 330.
   - 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`.
5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Example

The following is an example configuration to output data using the ODBC output engine.

```
[MyDatabase]
Type=odbc
OdbcConnectionString=DSN=MyDSN;
Mode=singlerecord
XslTemplate=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.

The example includes the following files:

- `configurations/examples/Other/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/examples/Other/BroadcastODBC.cfg` is an example configuration file that outputs data through ODBC. Before starting a `process` action with this configuration, find the `[ODBC]` section, and set the configuration parameter `ODBCConnectionString` to the connection string to use to connect to your database.
- `configurations/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 51: 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=HTTPpost
   ```

3. Create a new configuration section to contain the task settings, and 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*.

4. Configure how to combine records into HTTP requests. For information about how you can combine records, see Choose How to Output Data, on page 330.
   - 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`.

5. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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=toCFS.xsl
Mode=event
EventTrack=NewsSegment.Result
```

Chapter 52: 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, Micro Focus 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 to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=XProtect
   ```

3. Create a new configuration section to contain the task settings, and 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=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*.

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

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.

Part VIII: Advanced Configuration

This section includes advanced topics such as Media Server chaining.

- Chain Media Servers
- Schedule Actions in Media Server

Chapter 53: Chain Media Servers

This section describes how to send records to another Media Server, for further processing.

Introduction

Media Server can send records to another Media Server for further processing. For example, you can run face detection on one Media Server and then send the records describing detected faces to another Media Server that runs face recognition.

In this architecture, the Media Server that ingests the source media is referred to as the "upstream" server, and the Media Server that you send records to is referred to as the "downstream" or "remote" server.

This feature, called *chaining*, can be useful in cases where:

- The downstream server is equipped with hardware, such as a graphics card, that enables it to perform some types of analysis much faster than the upstream server(s). Chaining can help you make best use of your hardware.
- The downstream server has access to resources, such as face databases, that are not available to the upstream server(s).

Consider a scenario where you want to run face detection and recognition on the video recorded by ten cameras. You could have several Media Servers ingesting video and running face detection. You could then send all of the detected faces to a single Media Server that is equipped with a GPU and has access to your face recognition databases. Face recognition is much faster when performed with a GPU, so with this configuration you only need a single Media Server to perform the face recognition step.

> **NOTE:**
> Audio data cannot be transferred between Media Servers. This means that you cannot run analysis operations such as language identification, speaker identification, and speech-to-text on a downstream Media Server. The rolling buffer encoder cannot be used on a downstream Media Server, and other encoding engines can be used only if you set `AudioInput=none`.

Transferring data between Media Servers involves some overhead, so chaining performs best when the amount of data transferred between the servers is small and the downstream Media Server performs resource-intensive analysis tasks. In the previous scenario, the upstream Media Server sends images of detected faces to the downstream server so that they can be identified. You should configure each upstream server to send only the best frame for each detected face, rather than every frame in which a face appears. You should also crop the images before sending them, so that each image includes only a detected face and not the entire scene. This significantly reduces the amount of data that is transferred between the servers.

If your use case requires only a moderate amount of analysis to be performed by the downstream Media Server, or the downstream Media Server requires a large proportion of the data that is ingested upstream, it can be more efficient to do all of the analysis on a single server.

There are two ways to configure chaining: *one-way chaining* and *feedback chaining*. The following table describes the differences between these configurations.

Feature	One-way chaining	Feedback chaining
Licensing	A visual channel on the downstream server is required for each `process` action running upstream. (When you start processing on the upstream server, a processing session also begins on the downstream server. This remains active until all processing has finished).	A visual channel on the downstream server can handle requests from several `process` actions running upstream, but requests from upstream servers must wait if more requests are received than visual channels are available. (Each processing session on the downstream server processes a single record. A processing session exists on the downstream server only while a record is being processed).
Architecture	The upstream Media Server sends records to the downstream server, and processing continues on the downstream server. Information is not returned to the upstream server. Configure your output tasks on the downstream server. (If you configure a response output task on the downstream server, the information is returned to you by the upstream server because you do not receive a response from the downstream Media Server).	The upstream Media Server sends records to the downstream server, which performs analysis and returns the results so that processing can continue on the upstream server. Configure your output tasks on the upstream server.
	Media Server sends all records to the downstream server as soon as they are ready to be processed. Tracking and integration work as you would expect on a single server.	Media Server sends one record to the downstream server, and waits for it to be analyzed and the results to be returned before sending another. This means that records are considered to be independent. Feedback chaining is therefore unsuitable when analysis depends on tracking and integration (analyzing many frames to produce a single result). An example of an analysis task that depends on tracking and integration is

		number plate recognition (ANPR). You can use analysis engines on the downstream server that have tracking capabilities, but these capabilities provide no benefit and tracking results are not available.
	You can send multiple tracks to the downstream server.	You can send only one track to the downstream server.
Configuration	The upstream Media Server sends records downstream using the `Post` output engine. The downstream server receives them using the `Receive` ingest engine.	The upstream Media Server sends records downstream using the `RemoteAnalysis` analysis engine. The downstream server receives them using the `Receive` ingest engine.

To divide processing between two servers, you must create two configurations. The configuration for the downstream Media Server must be saved in the folder specified by the `ConfigDirectory` parameter in the `[Paths]` section of the configuration file (on the downstream Media Server that you are sending records to). You cannot include the configuration for the downstream server in the `process` action you send to the upstream server.

Configure One-Way Chaining

This section describes how to configure one-way chaining.

Configure the Upstream Media Server

This section describes how to configure Media Server to send records to another Media Server for further processing.

To send records to another Media Server

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `Engine`*`N`* parameter. You can give the task any name, for example:

   ```
   [Session]
   ...
   Engine3=FurtherProcessing
   ```

3. Create a new configuration section by typing the task name inside square brackets. Then, set the following parameters:

   `Type` The output engine to use. Set this parameter to **`Post`**.

`Input`	A comma-separated list of the tracks that you want to send to the downstream Media Server. You can provide aliases to simplify the track names when you configure the downstream Media Server. To do this specify the track names as *`Alias`*`:`*`TaskName`*`.`*`TrackName`*, where: • *`Alias`* is the track name to present to the downstream Media Server. • *`TaskName`* is the name of the task that produced the track. • *`TrackName`* is the name of the track. **NOTE:** You cannot send audio tracks to a downstream Media Server.
`Host`	The host name or IP address of the Media Server to send records to.
`Port`	The ACI port of the Media Server to send records to.
`ConfigName`	The name of the configuration file that the downstream Media Server must use to continue processing the records (this file must be present in the folder specified by the `ConfigDirectory` configuration parameter on the downstream Media Server).

For example:

```
[FurtherProcessing]
Type=Post
Input=FaceDetect.ResultWithSource
// or using alias "DetectedFaces" for "FaceDetect.ResultWithSource":
// Input=DetectedFaces:FaceDetect.ResultWithSource
Host=gpu-mediaserver
Port=14000
ConfigName=DownstreamFaceRec
```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Configure the Downstream Media Server

This section describes how to configure Media Server to receive records from another Media Server and continue processing them.

To receive records from another Media Server

1. Create a new configuration file, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `EngineN` parameter. You can give the task any name, for example:

```
[Session]
Engine0=RecordsFromUpstream
```

3. Create a new configuration section by typing the task name inside square brackets. Then, set the following parameters:

`Type`	The ingest engine to use. Set this parameter to **`Receive`**.
`Input`	A comma-separated list of tracks to receive from the other Media Server.

For example:

```
[RecordsFromUpstream]
Type=Receive
Input=FaceDetect.ResultWithSource
// or using alias "DetectedFaces", if defined in upstream configuration:
// Input=DetectedFaces
```

4. Configure the analysis, encoding, ESP, and output tasks that you want to run on the ingested records.

> **NOTE:**
> The tracks produced by the `Receive` engine are prefixed with the name of the ingest task. For example, if you use the configuration above the track produced by the `Receive` engine is named:
>
> - `RecordsFromUpstream.FaceDetect.ResultWithSource` (if no alias is defined).
> - `RecordsFromUpstream.DetectedFaces` (if the alias `DetectedFaces` is defined by the upstream Media Server).
>
> You must use the correct track name for the input of your face recognition task on the downstream Media Server.

5. Save the configuration file in the location specified by the `ConfigDirectory` parameter.

Example Configurations

This section includes example configurations that demonstrate how to chain two Media Servers.

The following configuration, for the upstream Media Server, runs face detection and then sends the records to a Media Server at `gpu-mediaserver:14000`:

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=FurtherProcessing

[Ingest]
Type=Video

[FaceDetect]
```

```
Type=FaceDetect
NumParallel=4
FaceDirection=Front
MinSize=10
SizeUnit=percent

[FurtherProcessing]
Type=Post
Input=FaceDetect.ResultWithSource
Host=gpu-mediaserver
Port=14000
ConfigName=DownstreamFaceRec
```

The following configuration, for the downstream Media Server, runs face recognition on records received from other Media Servers. To match the upstream configuration, above, this should be saved as `DownstreamFaceRec.cfg`, in the folder specified by the `ConfigDirectory` parameter on the downstream Media Server.

```
[Session]
Engine0=RecordsFromUpstream
Engine1=FaceRecognition
Engine2=IDOL

[RecordsFromUpstream]
Type=Receive
Input=FaceDetect.ResultWithSource

[FaceRecognition]
Type=FaceRecognize
Input=RecordsFromUpstream.FaceDetect.ResultWithSource
RecognitionThreshold=60
MaxRecognitionResults=1

[IDOL]
Type=IDOL
Input=FaceRecognition.Result
IdolHost=content
IdolPort=9000
IdolDB=BroadcastVideo
Mode=Time
OutputInterval=60
XslTemplate=toIDX.xsl
```

Start and Stop Processing

This section describes how to start and stop processing, when you have chosen to split processing between multiple Media Servers.

To start processing

- Start processing as described in Start Processing, on page 110. Send the `Process` action to the upstream Media Server only. You can use the `Config`, `ConfigName`, or `ConfigPath` action parameter to specify the configuration for the upstream Media Server.

  The upstream Media Server automatically starts a session on the downstream Media Server.

To stop processing

- Stop processing as described in Stop Processing, on page 113. Send the `QueueInfo` action to the upstream Media Server only.

  The upstream Media Server automatically stops the session on the downstream Media Server.

Configure the Maximum Number of Sessions

Your Media Server may receive many requests from upstream Media Servers. For example, you might have several Media Servers running face detection but use a single Media Server with a GPU to perform face recognition for all of your cameras or video feeds.

The maximum number of processing sessions to run concurrently (as a result of requests from upstream Media Servers) is configured by the `MaxProcessingSessions` parameter in the `[Chaining]` section of the Media Server configuration file:

```
[Chaining]
MaxProcessingSessions=1
QueueTimeout=60s
```

The `MaxProcessingSessions` parameter only limits sessions requested by upstream Media Servers; it has no effect on `process` actions sent directly to the Media Server (the number of `process` actions to run concurrently is controlled by the `MaximumThreads` parameter, as described in Process Multiple Requests Simultaneously, on page 85).

The default value of the `MaxProcessingSessions` parameter is `1`, so if you want to run more than one downstream session concurrently, you must increase the value of this parameter.

If the Media Server receives a greater number of requests than specified by `MaxProcessingSessions`, the additional requests are added to a queue and only start when other sessions finish. The upstream Media Server does not start ingesting the source media until the downstream Media Server is ready to start processing.

The `QueueTimeout` configuration parameter specifies the maximum amount of time that a request from an upstream Media Server can remain in the queue. If this timeout is exceeded then the request is removed from the queue and an error is returned to the upstream Media Server. If you are processing live streams, you might want to return an error to the upstream Media Server quickly.

Configure Feedback Chaining

This section describes how to configure feedback chaining.

Enable Feedback Chaining

You must enable feedback chaining by making some configuration changes on your downstream Media Server.

In the `[Chaining]` section of the configuration file, set the parameter `RemoteAnalysisConfigurations`. Specify a comma-separated list of configurations that can be used to perform remote analysis for other Media Servers, and the maximum number of processing sessions to allow concurrently for each configuration. For example:

```
RemoteAnalysisConfigurations=RemoteFaceRecognition:2,RemoteObjectDetection:3
```

This example permits Media Server to use the configuration files `RemoteFaceRecognition.cfg` and `RemoteObjectDetection.cfg` to perform analysis for other Media Servers.

An upstream Media Server could run two `process` actions that include remote face recognition, and three that include remote object recognition, and the downstream Media Server could handle requests from all of these actions simultaneously.

An upstream Media Server could run more than two `process` actions that include remote face recognition and more than `three` that include remote object recognition. In this case the requests for remote analysis might have to be queued on the downstream Media Server until previous requests have completed. However, the remote analysis tasks on the upstream server are unlikely to send every video frame downstream. Remote analysis requests that originated from one upstream `process` action can be processed while there are no requests being received from another. Each remote analysis request contains only a single record, so even if requests do need to be queued, the requests do not wait in the queue for very long.

The limit for the total number of connections to the downstream server is specified by the parameter `MaxRemoteAnalysisConnections`. The value that you choose for this parameter is the maximum number of `process` actions, that include remote analysis, that you can run concurrently on the upstream server(s).

You should also ensure that the downstream Media Server requests sufficient visual channels from your License Server. Media Server requires a visual channel for each remote analysis request that is processed concurrently. For the previous example, you would need to set `VisualChannels` to a value no less than `5`. The visual channels are required regardless of whether Media Server receives any requests.

The configuration file for the downstream Media Server might therefore contain the following:

```
[Channels]
VisualChannels=5

[Chaining]
RemoteAnalysisConfigurations=RemoteFaceRecognition:2,RemoteObjectDetection:3
MaxRemoteAnalysisConnections=10
```

The chaining configuration parameters `MaxProcessingSessions` and `QueueTimeout` do not apply to feedback chaining.

Configure the Upstream Media Server

This section describes how to configure Media Server to send records to another Media Server.

To send records to another Media Server

1. Create a new configuration to send to Media Server with the `process` action, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=Ingest
   Engine1=...
   Engine2=RemoteAnalysis
   ```
3. Create a new configuration section by typing the task name inside square brackets. Then, set the following parameters:

`Type`	The analysis engine to use. Set this parameter to **`RemoteAnalysis`**.
`Host`	The host name or IP address of the Media Server to send records to.
`Port`	The ACI port of the Media Server to send records to.
`ConfigName`	The name of the configuration file that the downstream Media Server must use to ingest and analyze the records (this file must be present in the folder specified by the `ConfigDirectory` configuration parameter on the downstream Media Server).
`Input`	The name of the track to send to the remote Media Server for remote analysis. You can provide an alias to simplify the track name when you configure the other Media Server. To do this specify the track name as *`Alias`*`:`*`TaskName`*`.`*`TrackName`*, where: • *`Alias`* is the track name to present to the remote Media Server. • *`TaskName`* is the name of the task that produced the track. • *`TrackName`* is the name of the track. **NOTE:** You cannot send audio tracks to a downstream Media Server.
`Output`	A comma-separated list of tracks, generated by the remote Media Server, to output from the remote analysis task. These tracks are then available for further processing on this Media Server.

Specify the track names as *Alias*:*TaskName*.*TrackName*, where:

- *Alias* is the track name to present to subsequent tasks on this Media Server.
- *TaskName* is the name of the task, on the remote Media Server, that produced the track.
- *TrackName* is the name of the track, on the remote Media Server.

For example:

```
[RemoteAnalysis]
Type=RemoteAnalysis
Host=gpu-mediaserver
Port=14000
ConfigName=RemoteFaceRecognition
Input=DetectedFaces:Crop.Output
// where "Crop" is a transform task to crop images of
// detected faces from the face detection ResultWithSource track
Output=RecognizedFaces:FaceRecognition.Result
// makes results available to subsequent tasks
// as RemoteAnalysis.RecognizedFaces
```

4. Save and close the configuration file. Micro Focus recommends that you save your configuration files in the location specified by the `ConfigDirectory` parameter.

Configure the Remote Media Server

This section describes how to create a configuration that ingests records from an upstream Media Server, and performs analysis on those records.

> **NOTE:**
> Before starting this procedure, enable feedback chaining by following the instructions in Enable Feedback Chaining, on page 368.

To receive records from another Media Server

1. Create a new configuration file, or open an existing configuration that you want to modify.
2. In the `[Session]` section, add a new task by setting the `EngineN` parameter. You can give the task any name, for example:

   ```
   [Session]
   Engine0=RecordsFromUpstream
   ```

3. Create a new configuration section by typing the task name inside square brackets. Then, set the following parameters:

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

`Input`	The name of the track to receive from the upstream Media Server.

For example:

```
[RecordsFromUpstream]
Type=Receive
Input=DetectedFaces
// where "DetectedFaces" is the alias specified by the Input parameter
// of the remote analysis task on the upstream server
```

4. Configure the analysis tasks that you want to run on the ingested records.

> **NOTE:**
> The tracks produced by the `Receive` engine are prefixed with the name of the ingest task. For example, if you use the configuration above the input track for your analysis task would be named `RecordsFromUpstream.DetectedFaces`.

> **NOTE:**
> Do not configure an output task. The records specified by the `Output` parameter in the remote analysis task on the upstream Media Server are returned to the upstream server automatically.

5. Save the configuration file in the location specified by the `ConfigDirectory` parameter.

Example Configurations

This section includes example configurations that demonstrate how to configure feedback chaining.

The following configuration, for the upstream Media Server, runs face detection and then sends records to a Media Server at `gpu-mediaserver:14000` for remote face recognition:

```
[Session]
Engine0=Ingest
Engine1=FaceDetect
Engine2=Crop
Engine3=RemoteAnalysis
Engine4=XML

[Ingest]
Type=Video

[FaceDetect]
Type=FaceDetect
FaceDirection=Front
MinSize=200
SizeUnit=pixel

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

```
[RemoteAnalysis]
Type=RemoteAnalysis
Host=gpu-mediaserver
Port=14000
ConfigName=RemoteFaceRecognition
Input=DetectedFaces:Crop.Output
Output=RecognizedFaces:FaceRecognition.Result

[XML]
Type=XML
Input=RemoteAnalysis.RecognizedFaces
XMLOutputPath=./output/html/%segment.type%_results_%segment.sequence%.html
XSLTemplate=toHTML.xsl
Mode=Time
OutputInterval=30s
```

The following configuration, for the remote Media Server, runs face recognition on records received from the upstream Media Server. To match the upstream configuration, above, this should be saved as `RemoteFaceRecognition.cfg`, in the folder specified by the `ConfigDirectory` parameter on the remote Media Server.

```
[Session]
Engine0=RecordsFromUpstream
Engine1=FaceRecognition

[RecordsFromUpstream]
Type=Receive
Input=DetectedFaces

[FaceRecognition]
Type=FaceRecognize
Input=RecordsFromUpstream.DetectedFaces
RecognitionThreshold=60
MaxRecognitionResults=1
```

Chapter 54: Schedule Actions in Media Server

This section describes how to schedule Media Server actions to run regularly.

Use IDOL Site Admin to Schedule Media Server Actions

The IDOL Site Admin user interface is an IDOL product that allows you to monitor and maintain your IDOL components from a central location. You can also use IDOL Site Admin to schedule actions in your IDOL components.

Set Up IDOL Site Admin to Monitor Media Server

The following section provides an overview of how to install and use IDOL Site Admin to monitor your Media Server systems. For more details about how to install and use IDOL Site Admin, refer to the IDOL Site Admin documentation.

To set up IDOL Site Admin

1. Install IDOL Site Admin and its prerequisite IDOL components. For details, refer to the *IDOL Site Admin Installation Guide*.
2. Start your IDOL Site Admin components, and then start the IDOL Site Admin service. You might need to perform some initial setup configuration. For details, refer to the *IDOL Site Admin Installation Guide*.
3. Log on to IDOL Site Admin.
4. On the top menu, click **Settings**, and then click **Settings**.

   The Settings page opens.
5. In the **Site Admin Application DB** section, select the database type. By default, IDOL Site Admin uses an embedded database. If you want to use a PostgreSQL database, you must perform additional setup steps. In this case, refer to the *IDOL Site Admin Installation Guide*.
6. Click the **Service Discovery Default Lookup Path** section to open it. This section enables you to set the paths on your hosts where you have IDOL components that you want to monitor, to allow IDOL Site Admin to discover services on those paths.

   Add all the paths that you use for the Media Servers that you want to use for scheduling.
7. Click **Save Changes** to save your configuration changes.

After you have installed and set up IDOL Site Admin, you can register your Media Server services to add them to IDOL Site Admin control.

To monitor Media Server

1. On each host machine that hosts a Media Server that you want to monitor and use for scheduling, install a Controller component.

   The Controller component monitors the IDOL components on a single host, and communicates with the Coordinator component, which in turn provides information to the IDOL Site Admin interface. For information about how to install Controller, refer to the *IDOL Site Admin Installation Guide*.

2. Start all your Media Servers and Controller components.
3. Log on to IDOL Site Admin.
4. Use the Discover **Hosts** page to find and register your Controllers. Register all the Controllers on the hosts you want to monitor. For more information, refer to the *IDOL Site Admin User Guide*.
5. Use the Discover **Services** page to find the installed IDOL components on each of your Controller hosts. Register all the Media Servers that you want to monitor. For more information, refer to the *IDOL Site Admin User Guide*.

Schedule Actions

After you have registered all your Media Servers, you can use the Monitor **Scheduler** page in IDOL Site Admin to add a schedule to run Media Server actions on your hosts. The scheduler sets up any ACI action to run on a regular schedule.

An IDOL Site Admin schedule runs a specified ACI action at a specified time, often with a recurrence schedule to repeat the action on a regular basis.

For more information about how to set up a schedule in IDOL Site Admin, refer to the *IDOL Site Admin User Guide*.

> **NOTE:**
> IDOL Site Admin does not limit the duration of an individual action. For Media Server actions that run on a video stream, you might need to set a maximum duration for the action.
>
> You can do this by setting the `MaximumDuration` parameter in either the video ingest engine configuration (see Ingest Video from a Stream, on page 126), or as an override parameter in the ACI action (see Override Configuration Parameters, on page 80).
>
> For example:
>
> ```
> action=process
> source=/video/sourcepath
> configname=myconfig
> [VideoStream]MaximumDuration=4h
> ```
>
> Setting this action in the IDOL Site Admin schedule processes the specified source file or stream with the configuration specified in the `myconfig` file. It overrides the value of `MaximumDuration` in the `[VideoStream]` configuration section to run the action for a maximum of four hours (where `[VideoStream]` is an ingest engine configuration with `Type=Video`).

Appendixes

This section contains the following appendixes.

- OCR Supported Languages
- OCR Supported Specialized Fonts
- ANPR Supported Locations
- Speech Analysis Supported Languages
- Pre-Trained Classifiers
- Pre-Trained Object Class Recognizers
- Encoding Profiles

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 - Australian Capital Territory	AU-ACT
Australia - New South Wales	AU-NSW
Australia - Northern Territory	AU-NT
Australia - Queensland	AU-QLD
Australia - South Australia	AU-SA
Australia - Tasmania	AU-TAS
Australia - Victoria	AU-VIC
Australia - Western Australia	AU-WA
Austria	AT
Bahrain	BH
Belarus	BY
Belgium	BE
Bosnia and Herzegovina	BA
Brazil	BR
Bulgaria	BG
Canada	CA
China	CN
Colombia	CO
Croatia	HR
Czech Republic	CZ

Denmark	DK
Estonia	EE
Finland	FI
France	FR
Georgia	GE
Germany	DE
Greece	GR
Hungary	HU
India	IN
Indonesia	ID
Ireland	IE
Israel	IL
Italy	IT
Japan	JP
Kingdom of Saudi Arabia	SA
Kuwait	KW
Latvia	LV
Lebanon	LB
Lithuania	LT
Macedonia	MK
Malaysia	MY
Mexico	MX
Moldova	MD
Montenegro	ME
Netherlands	NL
Nigeria	NG
Norway	NO
New Zealand	NZ

Oman	OM
Pakistan	PK
Peru	PE
Philippines	PH
Poland	PL
Portugal	PT
Qatar	QA
Romania	RO
Russia	RU
Serbia	RS
Singapore	SG
Slovakia	SK
Slovenia	SI
South Africa	ZA
Spain	ES
Sweden	SE
Switzerland	CH
Syria	SY
Thailand	TH
Tunisia	TN
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	GB
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: Speech Analysis Supported Languages

The following table describes the languages that are supported by language identification and speech-to-text. The "16kHz" columns refer to analysis for broadband audio and the "8kHz" columns refer to telephony.

Language		Language ID		Speech To Text	
Name	**Code**	**16kHz**	**8kHz**	**16kHz**	**8kHz**
Arabic - Gulf Arabic	ARGU			Y	
Arabic - Modern Standard Arabic	ARMSA	Y		Y	Y
Catalan	CAES			Y	Y
Chinese - Mandarin	ZHCN	Y		Y	Y
Czech	CSCZ		Y		Y
Danish	DADK	Y	Y	Y	Y
Dutch	NLNL	Y	Y	Y	Y
English - Australian	ENAU		Y	Y	Y
English - British	ENUK	Y	Y	Y	Y
English - Canadian	ENCA			Y	Y
English - Generic	ENXX			Y	
English - Irish	ENIE				Y
English - Singaporean	ENSG			Y	
English - South African	ENZA				Y
English - US	ENUS	Y	Y	Y	Y
Flemish	NLBE				Y
Farsi	FAIR	Y		Y	Y
French	FRFR	Y	Y	Y	Y
French - Canadian	FRCA		Y	Y	Y
German	DEDE	Y		Y	Y
Greek	ELGR	Y		Y	Y

Language		Language ID		Speech To Text	
Name	**Code**	**16kHz**	**8kHz**	**16kHz**	**8kHz**
Hebrew	HBIL	Y		Y	
Hindi	HIIN			Y	
Hungarian	HUHU			Y	Y
Italian	ITIT	Y	Y	Y	Y
Japanese	JAJP	Y		Y	Y
Korean	KOKR	Y		Y	Y
Polish	PLPL	Y	Y	Y	Y
Portuguese	PTPT			Y	
Portuguese - Brazilian	PTBR	Y		Y	Y
Romanian	RORO	Y	Y	Y	Y
Russian	RURU	Y	Y	Y	Y
Slovak	SKSK	Y		Y	Y
Spanish	ESES	Y	Y	Y	Y
Spanish - Latin American	ESLA	Y	Y	Y	Y
Spanish - North American	ESUS			Y	Y
Swedish	SVSE	Y	Y	Y	Y
Turkish	TRTR			Y	
Welsh	CYUK	Y			

Appendix E: Pre-Trained Classifiers

Micro Focus may provide classifiers that you can use with Media Server to classify images.

The following classifiers are currently available from the Big Data Download Center, in the package `PretrainedModels_MediaServer_<VERSION>.zip`. When you download this package, ensure that `<VERSION>` matches the version of Media Server that you are using.

File name	Description
`ImageClassifier_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.
`ImageClassifier_RoadScene.dat`	A classifier for classifying road scenes. This classifier can classify images into the classes "car", "person", and "van".

For information about how to import a classifier into your training database, see Import a Classifier, on page 174

Appendix F: Pre-Trained Object Class Recognizers

Micro Focus may provide pre-trained recognizers that you can use with Media Server to recognize objects belonging to generic categories in images and videos.

The following recognizers are currently available from the Big Data Download Center, in the package `PretrainedModels_MediaServer_<VERSION>.zip`. When you download this package, ensure that `<VERSION>` matches the version of Media Server that you are using.

File name	Description
`ObjectDetector_CommonObjects.dat`	Recognizes common objects. This recognizer contains twenty classes across four categories: • (Person) person • (Animal) bird, cat, cow, dog, horse, sheep • (Vehicle) aeroplane, bicycle, boat, bus, car, motorbike, train • (Indoor) bottle, chair, dining table, potted plant, sofa, tv/monitor
`ObjectDetector_HeadAndShoulder.dat`	Recognizes people, in order to count them. This recognizer differs from `ObjectDetector_Person.dat` because it has been trained to detect only the head and shoulder region, which is useful when you want to count people in a crowded area.
`ObjectDetector_Person.dat`	Recognizes people.
`ObjectDetector_RoadScene.dat`	Recognizes cars, vans and people in images and videos.

For information about how to import a recognizer into your training database, see Import a Recognizer, on page 180.

Appendix G: Encoding Profiles

This appendix describes the encoding profiles that are supplied with Media Server for use with the MPEG and Rolling Buffer encoders.

Video Profiles

The following profiles are MPEG4 profiles that use the H264 codec for encoding video. These are suitable for playback over the web, for example if your users view video using the MMAP Media Player or the native player of a web browser that supports H264.

Name	Codec	Variable Bitrate (kbps min-max)	Resolution	Display size	Notes
mpeg4video_h264_hd	H264	3000-8000	1920x1080	1920x1080	High-quality, high-definition.
mpeg4video_h264_720p	H264	800-3000	1280x720	1280x720	High-quality 1280x720.
mpeg4video_h264_360p	H264	100-800	640x360	640x360	High-quality 640x360.
mpeg4video_h264_sd	H264	800-3000	1024x576	1024x576	Use to encode video from a standard definition broadcast source that had non-square pixels.

The following profiles use the MPEG2 codec. Micro Focus recommends using the MPEG4 profiles (above) in most cases, but you can use these profiles if you need to support a media player that cannot decode H264.

Name	Codec	Variable Bitrate (kbps min-max)	Resolution	Display size	Notes
mpeg2video	MPEG2	1000-8000	720x576	720x576	Standard 4:3 MPEG2 video.
mpeg2video_broadcast	MPEG2	5000-13000	720x576	1024x576	Encodes with non-square pixels (the display size is not the same as the

					resolution), 16:9 aspect ratio, similar to DVB broadcast quality.

Audio Profiles

Name	Codec	Bitrate (kbps)	Sample rate (kHz)	Channels	Notes
mpeg4audio	MPEG4 AAC	128	48	2	Use with MPEG4 video profiles.
mpeg2audio	MPEG2 Audio	192	48	2	Use with MPEG2 video profiles.

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.

integration
When Media Server finds the same object (for example, the same number plate) across multiple video frames, integration aggregates the results to help filter out occasional outliers (for example, if one of the characters on the number plate is read incorrectly in one of the frames).

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. For more information, refer to the IDOL Document Security Administration Guide.

K

KeyView
The IDOL component that extracts data, including text, metadata, and subfiles from over 1,000 different file types. KeyView can also convert documents to HTML format for viewing in a Web browser.

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 Administration Guide (Micro Focus Media Server 12.1)

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 swpdl.idoldocsfeedback@microfocus.com.

We appreciate your feedback!