OpenVideo Guide
# Table of Contents

1 Introduction ........................................................................................................................................ 4
   1.1 Purpose ........................................................................................................................................ 4
   1.2 Scope ........................................................................................................................................ 4

2 Overview .......................................................................................................................................... 4

3 OpenVideo-Recorder API Overview .................................................................................................. 6
   3.1 Configuration Parameters ........................................................................................................... 6
      3.1.1 Configuration methods .......................................................................................................... 6
      3.1.2 Helper functions supplied with the template project .......................................................... 7
   3.2 Advanced Channel configuration ............................................................................................... 8
      3.2.1 The CAMERA_DATA4 Structure ....................................................................................... 8
      3.2.2 Camera Capabilities XML ............................................................................................... 12
      3.2.3 Additional settings for advanced channel configuration .................................................. 18
   3.3 Configuration Import ................................................................................................................... 19
      3.3.1 Obtain Camera Channel Count ......................................................................................... 21
      3.3.2 Retrieve the Camera IDs ..................................................................................................... 21
      3.3.3 Retrieve a Camera Based on Channel ID ......................................................................... 21
      3.3.4 Retrieve the number of PTZ presets a camera has ............................................................ 21
      3.3.5 Retrieve the PTZ presets information ............................................................................... 21
   3.4 Status and events ......................................................................................................................... 22
      3.4.1 Translator initialization........................................................................................................ 22
      3.4.2 Event synchronization ........................................................................................................ 23
      3.4.3 Status events .................................................................................................................... 23
      3.4.4 Firmware version events .................................................................................................... 24
      3.4.5 Communication Lost/Restored events ............................................................................ 24
      3.4.6 Other predefined events .................................................................................................. 25
      3.4.7 Generic text events .......................................................................................................... 25
   3.5 Video playback ............................................................................................................................ 26
      3.5.1 Translator initialization ........................................................................................................ 26
      3.5.2 Live Video playback .......................................................................................................... 28
      3.5.3 Recorded Video playback .................................................................................................. 28
      3.5.4 Turn off Cell Hiding ......................................................................................................... 28
      3.5.5 Playback control ............................................................................................................... 29
   3.6 Video Import and Export ............................................................................................................. 30
   3.7 Change password ......................................................................................................................... 30
      3.7.1 Change recorder password ............................................................................................... 31
      3.7.2 Change camera password ................................................................................................. 31
   3.8 Event recording ............................................................................................................................. 32
   3.9 Event locking ................................................................................................................................. 32
   3.10 PTZ Control ................................................................................................................................. 33
      3.10.1 Windows messages .............................................................................................................. 33
      3.10.2 Activation of the client (digital) PTZ control ..................................................................... 33
      3.10.3 Activation of the server (optical) PTZ control ................................................................... 34
      3.10.4 PTZ Control interface ....................................................................................................... 35
   3.11 Camera online/offline .................................................................................................................. 39
   3.12 Detect Recorder Type and Cameras ............................................................................................ 39
      3.12.1 Detect recorder type ......................................................................................................... 39
      3.12.2 Camera Wizard ............................................................................................................... 40
   3.13 Recorder Configuration ............................................................................................................... 40
3.13.1 Send alarm if unable to keep video on the recorder ................................................... 40
3.14 Archive Video ............................................................................................................. 41
  3.14.1 Event archiving ................................................................................................. 42
  3.14.2 Continuous archiving ....................................................................................... 42
3.15 Set Clock .................................................................................................................. 43
3.16 Dry Contacts (Inputs and Outputs) ............................................................................. 44
  3.16.1 Inputs/Outputs configuration in OnGuard ........................................................... 44
3.17 Failover and redundancy ........................................................................................... 45
  3.17.1 Overview ............................................................................................................ 45
  3.17.2 Additional implementation steps ........................................................................ 46
3.18 Launch recorder web page from OnGuard ............................................................... 50

4 Dynamic DVR Capabilities ............................................................................................ 51
  4.1 XML Schema Elements ........................................................................................... 51
  4.2 Sample XML Capabilities File .................................................................................. 53

5 OnGuard Installation and Configuration ....................................................................... 55
  5.1 Translator Usage ..................................................................................................... 55
  5.2 Documentation ......................................................................................................... 55
1 Introduction

1.1 Purpose

The purpose of this document is to provide a description of the OpenVideo-Recorder API which is one of the Open Alliance Program interfaces available for interfacing between OnGuard and third-party products. OpenVideo-Recorder is used in order to interface with third party digital video recorders. This document also describes all digital video features in OnGuard that can be used with interfaced recorders via OpenVideo-Recorder. In addition, this document provides the mapping between these features and the API functions that need to be implemented to enable these features.

1.2 Scope

The Lenel OpenAccess Alliance Program (OAAP) allows third-party manufacturers (OAAP partners) to interface their products with the OnGuard platform. In order to integrate a recorder, partners need to develop a translator DLL that implements a subset of COM interfaces published by Lenel for the OpenVideo-Recorder Application Programming Interface (API). OpenVideo-Recorder allows digital video recorders to be interfaced with OnGuard. For more information about the OAAP program and setting up the developer’s computer related to this integration, refer to the DeveloperGuide.pdf document.

2 Overview

OnGuard currently provides the following digital video features. Unless stated otherwise, the features below are supported in all versions of OnGuard.

1. Set Recorder in System Administration. As part of the OAAP certification process described in the DeveloperGuide.pdf document, the interfaced recorder type will be published in the OnGuard database under an official, unique name. This recorder type will be available in the Video Recorder Type drop-down list on the System Administration > Digital Video > Video Recorder form. Note that during translator development, before the translator project is “finalized” as described in the DeveloperGuide.pdf, the “Generic Video” recorder type needs to be selected in the drop-down list.

2. Recorder and channel configuration in System Administration. Until OnGuard 6.1.222 (including), the recorder and channel configuration was limited to basic recorder configuration and minimal channel configuration (name and id). This setup required the user to set the channels both in OnGuard and in the third party recorder.

   In OnGuard 6.3.249 or later, all features supported by OnGuard for channel configuration are available for OAAP channel configuration. Section 3.2.1 outlines the CAMERA_DATA4 structure which is used to send the channel configuration as it configured in the OnGuard user interface to the recorder. Section 3.2.2 explains how the “recorder’s capabilities” can be called from OnGuard making it possible for On Guard’s UI to support the OAAP channels features.

3. Channel configuration import.

4. Status of the recorder and/or recorder channels in Alarm Monitoring. OAAP integration will allow the status of the recorder and/or recorder channels to be reported in the OnGuard Alarm Monitoring application. The system hardware tree in Alarm Monitoring will show the recorders and channels configured via OnGuard System Administration in the same way as other devices in the system. Red and yellow “X” icons will be displayed on the recorder and channel devices that are offline.

5. Recorder sends events to OnGuard. The recorder can send out predefined and/or generic text events to OnGuard that will be handled according to the configuration for the corresponding event types. These events will be stored in the OnGuard database and optionally displayed in Alarm Monitoring.
These events (as well as any other OnGuard events) can also be configured to trigger Global I/O actions and other operations in the system that can be associated with events, such as event recording and launching video on event.

6. **Live and recorded video playback.** Alarm Monitoring and VideoViewer applications can be used to display live and/or recorded video for any video channel configured in System Administration. Also, in OnGuard 5.12.012 or later, live video can optionally be displayed from the System Administration’s Digital Video > Camera tab. Note that in order to take advantage of this support certain changes must be made to Accessory Add-ons to enable them to be installed on stand-alone Remote Monitor installations. Refer to the DeveloperGuide.pdf document for more information on Accessory Add-ons. Support for OAAP recorders for the VideoViewer browser-based client application is planned in a future release of OnGuard.

7. **Video Archive Server.** The archive server is supported for OAAP recorders in OnGuard 6.1 or later. The archive feature (along with file extension) should be enabled from the DVR Capabilities file in order to be supported.

8. **Video Export and Import.** Video Export and Import functionalities are available for OAAP recorders. An XML DVR Capabilities file must be implemented in order to enable the use of the video import feature.

9. **Event Recording.** SDK documentation for the existing timestamp-based event recording feature has not been published prior to this document; however the functionality exists in all currently supported versions of OnGuard. Support for this feature can be added retroactively to a translator DLL for any version of OnGuard. A new feature, as well being able to publish a new COM interface for it in the SDK documentation, called ID-based event recording is also planned. Currently, events being recorded are marked and later retrieved based on the timestamp of the event. The new type of event recording will use a unique event ID instead required by some customers.

10. **Event locking.** Locking video events to ensure the associated video does not get deleted from the recorder during recycling is an existing feature in OnGuard. This document describes the methods that need to be implemented in order to add this support.

11. **Recorder PTZ.** For OnGuard 5.12.012 or earlier, PTZ for OAAP recorders is only supported via the OnGuard built-in support for matrix switchers. No additional implementation is required to take advantage of this support. As of OnGuard 5.12.012, support has been added for recorder-based PTZ for OAAP recorders. This allows OAAP partners to provide their own implementation for PTZ commands. This document contains all the information about the PTZ interface.

12. **Intelligent Video.** The Lenel Intelligent Video Server can be used by OAAP recorders starting OnGuard 6.0.148. The implementation requirements can be found in the OpenVideo-Retrieval API.

13. **Recorder Configuration.** Some recorder configuration is available in OnGuard 6.1.222 or later – video resolution, video standard, delete old video files, detect cameras, and detect recorder types. All configurations need to be enabled from the DVR Capabilities file.

14. **Set Clock.** The set clock feature, which enables time synchronization between the OnGuard system and the OAAP recorder, is available for OAAP Recorders in OnGuard 6.1.222 or later. The feature needs to be enabled from the DVR Capabilities file.

15. **Dry Contacts (Inputs/Outputs).** In OnGuard 6.1.222 or later, a recorder’s dry contacts (inputs and outputs) can be integrated with the OnGuard system. This feature needs to be enabled from the DVR Capabilities file.
3 OpenVideo-Recorder API Overview

The OnGuard SDK disc contains the template translator project for each type of API, including OpenVideo-Recorder. The OpenVideo-Recorder template project is named GenericVideoTranslator. It is strongly recommended to use this project as the starting point for the translator development. As described in the sections below, the template project implements some common functionality that most OAAP partners need. It is also set to register the compiled translator DLL with the correct GUID that matches the “Generic Video” recorder type available in OnGuard that OAAP partners should use during initial development stage. As described in the DeveloperGuide.pdf, this GUID will need to be changed later during translator finalization.

IMPORTANT: The INC32 and LIB32 folders that come with the template project must be updated when switching development to another version of OnGuard. These folders should also be copied from the corresponding version of the OnGuard SDK disc and the translator project be recompiled against these even when no code changes are needed in the translator. Memory corruption issues are possible if this step is omitted.

Some OpenVideo COM interfaces are described in greater detail in the OpenDevice - InterfacesGuide.pdf document. Another method of organizing the API functions and related helper functions that the template project provides is described below. This document does not list all the functions of the API that are available. Most of the functions not listed are either obsolete or not commonly used. This document concentrates only on those functions that are typically used to add support for the features described.

3.1 Configuration Parameters

Configuration interfaces are used to pass configuration parameters entered into System Administration to the translator instance when it is created. The configuration methods that follow need to be implemented by the OAAP partner, followed by the helper functions that are already provided.

3.1.1 Configuration methods

The following methods need to be implemented for configuration purposes:

HRESULT Lnl_SetCamera(BYTE* pCameraData, DWORD cbSize);

This function belongs to the IVideoRecordConfig interface. It is called by the Communication Server when the end user adds or modifies a channel for the recorder or during a database download to the recorder. (This can be triggered by right-clicking on the recorder’s list item in the System Administration > Video > Digital Video > Video Recorder tab and selecting “Download.”). The passed pCameraData pointer contains parameters for the channel that were specified on the Camera tab in System Administration. The parameters are ordered in the CAMERA_DATA4 structure. (For more information, refer to section 3.2.1.) The pCameraData pointer should be cast to the CAMERA_DATA4 structure as illustrated below.

```
#include “lmsgtype.h”

HRESULT Lnl_SetCamera(BYTE* pCameraData, DWORD cbSize)
{
    if (cbSize < sizeof(CAMERA_DATA4))
        return E_FAIL;
    const CAMERA_DATA4* pData4 = reinterpret_cast<const CAMERA_DATA4*>(pCameraData);
    ...
```
The following lines should be added to the VideoTrans.h file (in addition to the IVideoRecorderConfig interface methods declarations):

To the declaration section:
#include "Interfaces\IVideoRecordConfig.h"

To the CVideoTrans declaration statement:
public IVideoRecordConfig,

To the BEGIN_COM_MAP statement:
COM_INTERFACE_ENTRY(IVideoRecorderConfig)

Typically the main purpose of the Lnl_SetCamera function is to update a list of channels currently configured in OnGuard for this recorder. It is recommended that the translator stores this list either locally or passes it to the recorder and then retrieves it back. One of the reasons for this list is to handle situations when not all of the physical channels on a recorder are configured in OnGuard for whatever reason. Events for channels should be filtered based on this list. In other words, if a channel is not configured in OnGuard, no events for that channel should be sent out to OnGuard.

HRESULT Lnl_DeleteCamera( CAMERA_DATA cameraData );

This function belongs to the IVideo interface. It is called by the Communication Server when an end user deletes a channel for this recorder. Typically this function is used by OAAP partners to update the channel list described above.

HRESULT SetCredentials( LPWSTR User, LPWSTR Domain, LPWSTR Password);

This method belongs to the new ILnrSetCredentials interface, which is available in OnGuard 5.12.012 or later. This interface is defined in the type library contained in the lnrapiu.h. This method is used when the UI_TYPE column of the PANELTYPE table in the OnGuard database is set to a value of 9 which enables the Username and Password UI controls in the Video Recorder tab in System Administration. In OnGuard 6.3.249 or later, the value of 6 in the UI_TPE column enables the user/password UI as well. This method passes the credentials stored in the OnGuard database each time the translator instance is created. Note that a bug in the original 5.12.012 release prevents this method from being called in some cases. OnGuard 5.12.012 Hot Fix 1.0 fixes this issue.

The following is an example of how to add support for the ILnrSetCredentials interface in the translator project. Simply add the following lines to the VideoTrans.h file:

To the declaration section:
#include "Ln1.Video.Api\lnrapi.h"

To the CVideoTrans declaration statement:
public ILnrSetCredentials,

To the BEGIN_COM_MAP statement:
COM_INTERFACE_ENTRY(ILnrSetCredentials)

3.1.2 Helper functions supplied with the template project
Some configuration methods are already implemented by the template GenericVideoTranslator project
and simply store passed parameters as member variables. Here are the helper functions (also supplied by the project) that can be used to retrieve these configuration parameters:

**GetPanelID()**

This function returns the unique panel ID for this recorder that is used internally in OnGuard. For example, the LNLMESSAGE ss_AccessPanelID field needs to be set to this value when sending events to OnGuard.

**GetPrimaryIP()**

This function returns the IP address of the recorder. When computer name option is used, the name stored in OnGuard database is resolved to a valid IP address prior to passing it to the translator instance. The IP address is stored as an integer. The four parts of the IP address are stored as follows: 32-25.24-17.16-9.8-1 – these numbers are bit order numbers, with least significant bit on the right.

**GetPanelTZInfo()**

This function returns the structure that contains time zone information for this recorder. This information, along with the daylight savings setting, is typically used to convert the time range of recorded video playback request from local to UTC (if needed).

**GetDST()**

This function returns the Boolean flag that indicates whether daylight saving time was checked for the recorder’s time zone.

### 3.2 Advanced Channel configuration

A more advanced channel configuration is available for OAAP recorders in OnGuard 6.3.249 or later. In order to enable advanced configuration, the following steps need to be implemented:

1) Using the Lnl_SetCamera method, get the channel configuration and send it to the recorder (For more information, refer to section 3.2.1)

2) Using the Lnl_UpdateLNRCaps method, send the capabilities of all cameras supported by the DVR (For more information, refer to section 3.2.2).

**Notes:**

The advanced configuration is only available when the UI_TYPE column in the PANELTYPE table is set to 6 for the OAAP recorder.

When UI_TYPE=6 is used, implementation of the Lnl_UpdateLNRCaps is mandatory.

#### 3.2.1 The CAMERA_DATA4 Structure

The camera configuration data is currently passed around in the form of the CAMERA_DATA4 structure. The structure is defined in the lmsgtype.h file. The configuration of the camera via the UI is saved to the database and loaded into the CAMERA_DATA4 when passed through the API to the device translators. Most of the properties of the data structure can be utilized for the recorder integration. However, some of the properties are marked as reserved.

As mentioned in section 3.1.1, the Lnl_SetCamera method (from the IVideoRecorderConfig interface)
is called when a channel is added or modified or when a download operation is made. The implementation of getting the CAMERA_DATA4 structure and handling it should be done in this method.

While each CAMERA_DATA4 field can be used, many of the fields depend on the camera capabilities passed to us by the OAAP video recorder (For more information, refer to section 3.2.2). For example, if the recorder specifies that brightness is not a feature supported by the camera, the brightness property in the structure is still filled in (it defaults to 0 in this case), it has no meaning since that feature is not supported and was not configured via the UI.

The following table describes each parameter of the CAMERA_DATA4 structure as well as its supporting camera capability as defined in section 3.2.2 Camera Capabilities XML.

The time zones can have any of the following values: -1=Recorder Configured Schedule, 1=Never, 2=Always.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
<th>CAMERA CAPABILITIES XML NODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerID</td>
<td>long</td>
<td>The database ID of the recorder</td>
<td></td>
</tr>
<tr>
<td>CameraID</td>
<td>long</td>
<td>The database ID of the camera</td>
<td></td>
</tr>
<tr>
<td>Channel</td>
<td>long</td>
<td>The recorder’s channel number for this camera</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>wchar_t[32]</td>
<td>The user defined name of the camera</td>
<td></td>
</tr>
<tr>
<td>cdEventPostroll</td>
<td>long</td>
<td>The amount of video in seconds to record after the event state was exited</td>
<td></td>
</tr>
<tr>
<td>Bright</td>
<td>long</td>
<td>RANGE(1-100) - How much to adjust the video for brightness of the picture</td>
<td>VideoSource &gt; Brightness</td>
</tr>
<tr>
<td>Contrast</td>
<td>long</td>
<td>RANGE(1-100) - How much to adjust the video for contrast between light and dark</td>
<td>VideoSource &gt; Contrast</td>
</tr>
<tr>
<td>Color</td>
<td>long</td>
<td>RANGE(1-100) - How much to adjust the intensity or removal of color of the picture</td>
<td>VideoSource &gt; Saturation</td>
</tr>
<tr>
<td>Hue</td>
<td>long</td>
<td>RANGE(1-100) - How much to shift the hue from a warmer color to a colder color</td>
<td>VideoSource &gt; Hue</td>
</tr>
<tr>
<td>FrameRate</td>
<td>long</td>
<td>The number of milliseconds per frame in the video stream</td>
<td>VideoSource &gt; FrameRates</td>
</tr>
<tr>
<td>IntraFrameRate</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>MotionBitRate</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>NonMotionBitRate</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>InMotionLevel</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>PreRoll</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>PostRoll</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>PARAMETER</td>
<td>TYPE</td>
<td>DESCRIPTION</td>
<td>CAMERA CAPABILITIES XML NODE</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>State</td>
<td>long</td>
<td>Returns the status of the camera from the recorder</td>
<td></td>
</tr>
<tr>
<td>MotionDetection</td>
<td>BYTE</td>
<td>BOOLEAN – If true, displays motion detection alarms in alarm monitoring</td>
<td>MotionDetection</td>
</tr>
<tr>
<td>MotionTimeZone</td>
<td>long</td>
<td>The timezone by which detection alarms are reported to alarm monitoring</td>
<td>MotionDetection</td>
</tr>
<tr>
<td>MotionMask</td>
<td>BYTE[32][24]</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>ContArchive</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>ContArchiveTimeZone</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>Quality</td>
<td>long</td>
<td>RANGE(1-100) – The quality of the compression (1 – lowest compression, 100 – highest)</td>
<td>VideoSource &gt; Quality</td>
</tr>
<tr>
<td>TimeLapseRecordMode</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>SurveillanceOnly</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>IPAddress</td>
<td>long</td>
<td>The IP address of the camera</td>
<td></td>
</tr>
<tr>
<td>Online</td>
<td>BYTE</td>
<td>BOOLEAN – Sets the camera as online (TRUE) or offline (FALSE)</td>
<td></td>
</tr>
<tr>
<td>cpPortNumber</td>
<td>DWORD</td>
<td>The port the connection to the camera is made</td>
<td>VideoSource &gt; Port</td>
</tr>
<tr>
<td>cdStaticTimeZone</td>
<td>DWORD</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cdCriteriaCount</td>
<td>DWORD</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cdBlingCamTimeZone</td>
<td>DWORD</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>CameraGUID</td>
<td>CLSID</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>CameraType</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>UserName</td>
<td>wchar_t[32]</td>
<td>The username used to login to the camera</td>
<td>VideoSource &gt; UserName</td>
</tr>
<tr>
<td>Password</td>
<td>wchar_t[32]</td>
<td>The encoded password used to login to the camera</td>
<td>VideoSource &gt; Password</td>
</tr>
<tr>
<td>Resolution</td>
<td>SIZE</td>
<td>Upper WORD is the width, the lower WORD is the height</td>
<td>VideoSource &gt; Resolutions</td>
</tr>
<tr>
<td>Rotation</td>
<td>short</td>
<td>The rotation of the video (in degrees)</td>
<td>VideoSource &gt; Rotation</td>
</tr>
<tr>
<td>DisplayTitle</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>DisplayDate</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>DisplayTime</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>Monochrome</td>
<td>long</td>
<td>BOOLEAN – Display non-color video (TRUE) or display color (FALSE)</td>
<td>VideoSource &gt; Monochrome</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>TYPE</td>
<td>DESCRIPTION</td>
<td>CAMERA CAPABILITIES</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Sharpness</td>
<td>long</td>
<td>RANGE (1-100) - The edge contrast level of the video picture. The greater the sharpness the higher the edge contrast</td>
<td>VideoSource &gt; Sharpness</td>
</tr>
<tr>
<td>Gamma</td>
<td>long</td>
<td>RANGE (1-100) – Corrects the light intensity along the luminous spectrum from darker to lighter</td>
<td>VideoSource &gt; Gamma</td>
</tr>
<tr>
<td>WhiteBalance</td>
<td>long</td>
<td>RANGE (1-100) – A color temperature adjustment, from warm to cold, that allows the most closely match with pure white</td>
<td>VideoSource &gt; WhiteBalance</td>
</tr>
<tr>
<td>BacklightComp</td>
<td>long</td>
<td>RANGE (1-100) - The amount of compensation for a large amount of background light</td>
<td>VideoSource &gt; BacklightComp</td>
</tr>
<tr>
<td>Exposure</td>
<td>long</td>
<td>RANGE (1-100) – The amount of time per frame the image is taken. A higher exposure will result in brighter but blurrier images</td>
<td>VideoSource &gt; Exposure</td>
</tr>
<tr>
<td>Iris</td>
<td>long</td>
<td>The size of the camera lens aperture – the larger, the more light will be allowed to enter</td>
<td>VideoSource &gt; Iris</td>
</tr>
<tr>
<td>SourceName</td>
<td>wchar_t[32]</td>
<td>Camera model</td>
<td>VideoSource &gt; SourceName</td>
</tr>
<tr>
<td>Standard</td>
<td>wchar_t[32]</td>
<td>The video broadcasting standard (i.e. NTSC)</td>
<td>VideoSource &gt; Standard</td>
</tr>
<tr>
<td>InputNumber</td>
<td>long</td>
<td>The input number of the camera to obtain the video</td>
<td>VideoSource &gt; InputNumber</td>
</tr>
<tr>
<td>EventFrameRateMs</td>
<td>long</td>
<td>The framerate in milliseconds per frame to record during an event</td>
<td>VideoSource &gt; EventFrameRateMs</td>
</tr>
<tr>
<td>EventResolution</td>
<td>long</td>
<td>RESERVED</td>
<td>VideoSource &gt; EventResolution</td>
</tr>
<tr>
<td>EventQuality</td>
<td>long</td>
<td>RESERVED</td>
<td>VideoSource &gt; EventQuality</td>
</tr>
<tr>
<td>EventPreroll</td>
<td>long</td>
<td>The amount of video in seconds to record before the event state was entered</td>
<td>VideoSource &gt; EventPreroll</td>
</tr>
<tr>
<td>LiveFrameRate</td>
<td>long</td>
<td>RESERVED</td>
<td>VideoSource &gt; LiveFrameRate</td>
</tr>
<tr>
<td>AudioChannel</td>
<td>long</td>
<td>ID – The channel which will be the input for the audio for this camera</td>
<td>VideoSource &gt; AudioChannel</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>TYPE</td>
<td>DESCRIPTION</td>
<td>CAMERA CAPABILITIES XML NODE</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>cd4Bitrate</td>
<td>long</td>
<td>The size of the bandwidth the camera will use (how many bits per seconds the video stream will be)</td>
<td>VideoSource &gt; Bitrate</td>
</tr>
<tr>
<td>btMotionTriggersEventRecording</td>
<td>BYTE</td>
<td>BOOLEAN – Determines if motion events will trigger event recording</td>
<td></td>
</tr>
<tr>
<td>cd4AudioVolume</td>
<td>long</td>
<td>The audio input volume amount</td>
<td>AudioSource2 &gt; AudioInVolume</td>
</tr>
<tr>
<td>cd4AudioSource</td>
<td>long</td>
<td>The audio input source type</td>
<td>AudioSource2 &gt; AudioInSource</td>
</tr>
<tr>
<td>cdAudioRecording</td>
<td>BYTE</td>
<td>BOOLEAN – If audio recording is enabled</td>
<td></td>
</tr>
<tr>
<td>cdEventAudioRecording</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cdRecordOnEvent</td>
<td>BYTE</td>
<td>BOOLEAN – If the event recording is enabled (TRUE) or disabled (FALSE)</td>
<td>CaptureInEventModeOnly</td>
</tr>
<tr>
<td>cd4UseCameraTimeStamps</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cd4OnCameraRecordingActivated</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cd4OnCameraRecordingTimeZone</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cd4FTPPortNumber</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cd4FTPRoot</td>
<td>wchar_t[32]</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cd4RecordingTimeZone</td>
<td>long</td>
<td>The timezone by which the recorder should record the video from this camera</td>
<td></td>
</tr>
<tr>
<td>cd4AllowAudioIntercom</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cd4AllowDirectConnect</td>
<td>BYTE</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cd4DirectConnectTimeout</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
<tr>
<td>cd4RTPProtocol</td>
<td>long</td>
<td>The protocol by which RTP will communicate</td>
<td>VideoSource &gt; RTP</td>
</tr>
<tr>
<td>cd4RTSPort</td>
<td>long</td>
<td>The port on which RTSP will connect</td>
<td>VideoSource &gt; RTP</td>
</tr>
<tr>
<td>cd4Redundant</td>
<td>long</td>
<td>RESERVED</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2.2  Camera Capabilities XML

To specify the capabilities of cameras for OnGuard, an XML capabilities specification must be created for each camera. This XML instructs the UI as to what can and cannot be configured for the camera and it therefore makes the UI dynamic and powerful. For more detail on what is allowed in the XML, the `LNRAPI.XSD` file which defines the XML schema, can be analyzed and should be used to validate the correctness of all finalized XML capabilities files.

The nodes defined in the table below (section 3.2.2.1) are the ones used for the OAAP camera abilities to be defined. At the base is the `<SourceCapabilities>` node (the root node). The table describes each specification node, its properties, and the reason to specify it. An example is included below (see section 3.2.2.2).
HRESULT Lnl_UpdateLNRCaps(BSTR* xml);

This function belongs to the IRecorderCaps2 interface. It is called by the System Administration
application when the OnGuard end user presses the Update Capabilities button
(Video>DigitalVideo>VideoRecorder tab> Update Capabilities button).
The method should set the cameras’ capabilities in the ‘xml’ variable.
When implementing the method, the IRecorderCaps interface methods need to be implemented as well
as the IRecorderCaps2 derives from IRecorderCaps. All the IRecorderCaps methods should return
E_NOTIMPL. The two other methods of the IRecorderCaps2 should return E_NOTIMPL as well.

In order to add support for the capabilities functionality the following lines should be added to the
VideoTrans.h file (in addition to the IRecorderCaps and the IRecorderCaps2 interfaces methods
declarations):

To the declaration section:
#include "Interfaces\IVideoRecordConfig.h"

To the CVideoTrans declaration statement:
public IRecorderCaps2,

To the BEGIN_COM_MAP statement:
COM_INTERFACE_ENTRY(IRecorderCaps)
COM_INTERFACE_ENTRY(IRecorderCaps2)

3.2.2.1 Table Specification

The table is laid out like a tree by indentation. If a node can have sub-nodes, they will be directly
below that node and indented further. All nodes at the same indentation level are siblings and cannot
be contained within each other but only within the parent node.

In the “Ocours” column, the range of the amount of possible occurrences is shown. For example,
“0-N” means that this node is not required to be specified but up to an infinite number can be specified.
“1” means that one and only one node must be specified. An “(E)” means that at least one sibling is
required.

Note that camera input and camera outputs are not supported for OAAP integration and should not be
included in the XML file.

<table>
<thead>
<tr>
<th>NODE</th>
<th>TYPE</th>
<th>OCCURS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceCapabilities</td>
<td>Root</td>
<td>1</td>
<td>The root of all capabilities</td>
</tr>
<tr>
<td>VideoSource</td>
<td>SubRoot</td>
<td>1</td>
<td>The root for video source (camera) capabilities</td>
</tr>
<tr>
<td>Brightness</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the brightness settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>Contrast</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the contrast settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>Saturation</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the saturation settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>Hue</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the hue settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>FrameRates</td>
<td>List/Range</td>
<td>0-1</td>
<td>Frame rate definition that can be in either the form of specified rates or ranges</td>
</tr>
<tr>
<td>NODE</td>
<td>TYPE</td>
<td>OCCURS</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FrameRate</td>
<td>Specification</td>
<td>0-N (E)</td>
<td>A specified frame rate for a particular Video Standard</td>
</tr>
<tr>
<td>FrameRateRange</td>
<td>Range</td>
<td>0-N (E)</td>
<td>A range of frame rates for a particular Video Standard</td>
</tr>
<tr>
<td>Quality</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the compression settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>Port</td>
<td>Specification</td>
<td>0-1</td>
<td>Specifies the default port of the camera and if this value can be changed</td>
</tr>
<tr>
<td>UserName</td>
<td>Boolean</td>
<td>0-1</td>
<td>Determines if the username to connect to the camera can be configured and is supported</td>
</tr>
<tr>
<td>Password</td>
<td>Boolean</td>
<td>0-1</td>
<td>Determines if the password to connect to the camera can be configured and is supported</td>
</tr>
<tr>
<td>Resolutions</td>
<td>SubRoot</td>
<td>1</td>
<td>Specifies a list and ranges of resolutions the camera supports</td>
</tr>
<tr>
<td>Resolution</td>
<td>Specification</td>
<td>0-N (E)</td>
<td>A specified resolution the camera supports</td>
</tr>
<tr>
<td>ResolutionRange</td>
<td>Range</td>
<td>0-N (E)</td>
<td>A range of resolutions the camera supports</td>
</tr>
<tr>
<td>Rotation</td>
<td>Range</td>
<td>0-1</td>
<td>Determines the video rotation settings</td>
</tr>
<tr>
<td>Monochrome</td>
<td>Boolean</td>
<td>0-1</td>
<td>Determines if the camera supports a monochrome mode</td>
</tr>
<tr>
<td>Sharpness</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the sharpness settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>Gamma</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the gamma settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>WhiteBalance</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the white-balance settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>BacklightComp</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the backlight compensation settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>Exposure</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the exposure settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>VideoStandards</td>
<td>SubRoot</td>
<td>0-1</td>
<td>A list of the video standards the camera supports (i.e. NTSC)</td>
</tr>
<tr>
<td>NODE</td>
<td>TYPE</td>
<td>OCCURS</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VideoStandard</td>
<td>Specification</td>
<td>1-N</td>
<td>A specification of a single video standard the camera supports</td>
</tr>
<tr>
<td>InputNum</td>
<td>Specification</td>
<td>0-1</td>
<td>A specification of the number of inputs the camera supports</td>
</tr>
<tr>
<td>Bitrate</td>
<td>List/Range</td>
<td>0-1</td>
<td>A list or range of bitrates the camera can stream</td>
</tr>
<tr>
<td>RTP</td>
<td>SubRoot</td>
<td>0-1</td>
<td>Determines if RTP is supported by this camera</td>
</tr>
<tr>
<td>RTSPPort</td>
<td>Specification</td>
<td>0-1 (E)</td>
<td>A specification of the default RTSP port</td>
</tr>
<tr>
<td>Protocol</td>
<td>List</td>
<td>0-1 (E)</td>
<td>A list of RTP protocols supported</td>
</tr>
<tr>
<td>AudioSource2</td>
<td>SubRoot</td>
<td>0-1</td>
<td>Determines if this camera supports audio recording</td>
</tr>
<tr>
<td>AudioInVolume</td>
<td>List/Range</td>
<td>0-1</td>
<td>Determines the volume settings (currently limited to a range from 1 to 100)</td>
</tr>
<tr>
<td>AudioInSource</td>
<td>List</td>
<td>0-1</td>
<td>The source(s) of the audio for this camera</td>
</tr>
<tr>
<td>CaptureInEventModeOnly</td>
<td>SubRoot</td>
<td>0-1</td>
<td>Determines if the video recorder can capture the video from this camera during an event</td>
</tr>
<tr>
<td>MaxPrerollMS</td>
<td>Specification</td>
<td>0-1</td>
<td>A specification of the maximum pre-roll in milliseconds</td>
</tr>
<tr>
<td>MotionDetection</td>
<td>Boolean</td>
<td>0-1</td>
<td>Determines if motion detection is supported by the channel (either the video recorder or camera for this camera)</td>
</tr>
<tr>
<td>ChangePasswordCapability</td>
<td>SubRoot</td>
<td>0-1</td>
<td>Determines if the video recorder's password for this camera can be changed (different from the password to the camera itself)</td>
</tr>
<tr>
<td>MinLength</td>
<td>Specification</td>
<td>0-1</td>
<td>Specifies the minimum length of the password</td>
</tr>
<tr>
<td>MaxLength</td>
<td>Specification</td>
<td>0-1</td>
<td>Specifies the maximum length of the password</td>
</tr>
<tr>
<td>CharSet</td>
<td>Specification</td>
<td>0-1</td>
<td>Specifies the allowable characters the password can be made of</td>
</tr>
</tbody>
</table>

3.2.2.2 Capabilities XML Example

This sample XML file provides a good starting point for creating a custom capabilities XML specification. Of course, if any options are not supported by the camera, they should be removed from the XML.
It is a good idea to play with the values and see how they affect the UI in System Administration, and then tweak them until they produce exactly what is intended and supported by the camera.

```xml
<SourceCapabilities type="static">
  <VideoSource>
    <VideoStandards>
      <VideoStandard>NTSC</VideoStandard>
      <VideoStandard>PAL</VideoStandard>
    </VideoStandards>
    <Port default="4001"/>
    <InputNum>5</InputNum>
    <UserName/>
    <Password/>
    <Resolutions>
      <Resolution height="640" width="480" VideoStandard="NTSC" default="1"/>
      <Resolution height="320" width="240" VideoStandard="NTSC"/>
      <Resolution height="640" width="480" VideoStandard="PAL" default="1"/>
      <Resolution height="320" width="240" VideoStandard="PAL"/>
    </Resolutions>
    <Quality>
      <Default>Auto</Default>
      <Manual max="100" step="10" min="10">
        <DiscreteValue>10</DiscreteValue>
        <DiscreteValue>20</DiscreteValue>
        <DiscreteValue>30</DiscreteValue>
        <DiscreteValue>40</DiscreteValue>
        <DiscreteValue>50</DiscreteValue>
        <DiscreteValue>60</DiscreteValue>
        <DiscreteValue>70</DiscreteValue>
        <DiscreteValue>80</DiscreteValue>
        <DiscreteValue>90</DiscreteValue>
        <DiscreteValue>100</DiscreteValue>
      </Manual>
      <Auto tag="Automatic"/>
    </Quality>
    <FrameRates>
      <FrameRateRange VideoStandard="NTSC" min="1" max="30" default="1"/>
      <FrameRateRange VideoStandard="PAL" min="1" max="25"/>
      <FrameRate units="fps" default="true" rate="15" VideoStandard="NTSC"/>
      <FrameRate units="fps" rate="30" VideoStandard="NTSC"/>
    </FrameRates>
    <Rotation max="270" step="90" min="0"/>
    <Brightness>
      <Default>50</Default>
      <Manual max="100" step="2" min="0"/>
    </Brightness>
    <Contrast>
      <Default>50</Default>
      <Manual max="100" min="0"/>
    </Contrast>
    <Hue>
      <Default>50</Default>
      <Manual max="100" min="0"/>
    </Hue>
    <Saturation>
      <Default>50</Default>
      <Manual max="100" min="0"/>
    </Saturation>
    <Sharpness>
      <Default>50</Default>
      <Manual max="100" min="0"/>
    </Sharpness>
  </VideoSource>
</SourceCapabilities>
```
3.2.2.3 Camera Type Name

In order to add the camera type name, add the following to the beginning of the XML:

```
<Sources xmlns="http://www.lenel.com/LNR/2002/LNRAPI">
    <Source>
        <Name>CameraTypename</Name>
    </Source>
</Sources>
```

By default, MJPEG codec is the only choice for camera type’s supported codecs. In order to support multiple codecs, a new camera type capability needs to be created for each additional codec. The name of the camera type needs to be exactly the same as the original name plus a space character followed by “MPEG4” or “H264.” For example, rename “CameraType1” for MJPEG codec “CameraType1 MPEG4”, and “CameraType1 264” for H264 codec.

3.2.3 Additional settings for advanced channel configuration

The following settings are available only when advanced channel configuration is implemented. (For more information, refer to section 3.2.)

3.2.3.1 Enable live settings

To enable the live frame rate UI in the Camera->Normal Mode tab, the `<LiveSettings/>` element should be added to the DVR Capabilities file. (For more information on the DVR capabilities file, refer to section 4 Dynamic DVR Capabilities.)

3.2.3.2 Limit recorded video frame rate by live frame rate

By default, any valid frame rate can be set for the camera’s recorded frame rate, regardless of the live frame rate selection. Adding the `<LimitRecordedFrameRateByLive/>` element to the DVR Capabilities xml will prevent users from specifying a recorded video frame rate that exceeds the currently selected live frame rate.

3.2.3.3 Set Sequential Recording Frame Rate

By default, OnGuard only allows valid live frame rates to be selected as recording frame rates – these frames are the ones specified via the `<FrameRates>` element in the camera type capability xml.

There is a method to allow all sequential frame rates in the valid range for the recording frame rate value. Using the `<SequentialRecordingFrameRates/>` element in the DVR Capabilities xml enables sequential values from 1 to max live frame rate to populate on the recording frame rate combo box (or from 1 to the currently selected live frame rate if the `<LimitRecordedFrameRateByLive/>` element is also specified as described in section 3.2.3.1).

For example, a camera might support only live frame rates of 10 and 15 fps. By default only 10 and 15 will be available as the recorded frame rate. Using the `<SequentialRecordingFrameRates/>` element, all frame rates from 1 - 15 will show up as a valid configuration.

3.2.3.4 Set time lapse recording

By default, an edit box is available for setting the time lapse recording value. The edit box allows the
OnGuard user to set any value.

There is a way to limit the supported time lapse values to a specific set of values, and to present them in a combo box. It is done on a per-camera-type basis by specifying individual `<FrameRate/>` elements in the camera type’s capabilities xml, and setting the “units” attribute value to “milliseconds.” A sample xml follows that defines both regular and time lapse frame rates:

```xml
<SourceCapabilities>
  ...
  <VideoSource>
    ...
    <FrameRates>
      <FrameRateRange min="1" max="30" default="30" units="fps"/>
      <FrameRate rate="40" units="fps"/>
      <FrameRate rate="45" units="fps"/>
      <FrameRate rate="2000" units="milliseconds"/>
      <FrameRate rate="4000" units="milliseconds"/>
      <FrameRate rate="5000" units="milliseconds"/>
      <FrameRate rate="7000" units="milliseconds"/>
      <FrameRate rate="12000" units="milliseconds"/>
    </FrameRates>
  </VideoSource>
</SourceCapabilities>
```

3.2.3.5 Motion Detection
To enable the motion detection UI in the channel configuration, the `<Motion Detection/>` element needs to be added to the Recorder Capabilities xml file.

3.2.3.6 Configure recorder time zones for Archive/Recording/Motion Detection
Currently OnGuard only supports three (3) hardcoded values for all time zones for OAAP channels (Always, Never, and `<Use Recorder Configuration>`). The `<Use Recorder Configuration>` allows the user to use the recorder’s own method of scheduling (that recorders may or may not allow via their own configuration UI).

3.2.3.7 Camera Direct Connect
To enable the direct connect UI (under the Camera->Normal Mode tab), the `<CameraDirectConnect/>` element needs to be added to the DVR Capabilities file.

3.2.3.8 Camera time stamps
To enable the camera time stamps UI (under the Camera->Normal Mode tab), the `<CameraTimeStamps/>` element needs to be added to the DVR Capabilities file.

3.2.3.9 Two way Audio
To enable the two-way audio UI (under the Camera->Audio tab), the `<TwoWayAudio/>` element needs to be added to the DVR Capabilities file.

3.3 Configuration Import
The primary reason for having the XML capabilities and CAMERA_DATA4 structure is to allow configuration of the camera channels via System Administration. As an added feature, the ability to obtain these configurations from the recorder has been added to simplify the integration of an existing video recorder into the OnGuard system.
In order to enable the **Import From Recorder** menu item in the context menu of the recorder (System Administration->Digital Video->Recorder tab), the `<ImportCameraConfig/>` element needs to be added to the DVR Capabilities file. (Refer to section 4 for more information about the DVR capabilities). Optionally, starting with OnGuard 6.5, the `<ImportPTZPresets/>` element can be added to the DVR capabilities file in order to enable import of the recorder’s PTZ presets along with the information about the configured channels.

**Note:** The configuration import is only available when the UI_TYPE column in the PANELTYPE table is set to 6 or 9 for the OAAP recorder.

The **IVideoCameraConfigImport** interface was added to accomplish the camera channel import functionality. It is comprised of three functions as described below (sections 3.3.1, 3.3.2, and 3.3.3). In order to support the interface, the following has to be added to the project:

- To the declaration section:
  ```
  #include "Interfaces\IVideoCameraConfigImport.h"
  ```

- To the CVideoTrans declaration statement:
  ```
  public IVideoCameraConfigImport,
  ```

- To the BEGIN_COM_MAP statement:
  ```
  COM_INTERFACE_ENTRY(IVideoCameraConfigImport)
  ```

The **IVideoPTZPresetImport** interface was added to enable import of pre-configured presets information at the recorder. It is not mandatory to implement this interface for the import functionality to work. The interface has two functions as described below (sections 3.3.4 and 3.3.5). In order to support the interface, the following must be added to the project:

- To the declaration section:
  ```
  #include "Interfaces\IVideoPTZPresetImport.h"
  ```

- To the CVideoTrans declaration statement:
  ```
  public IVideoPTZPresetImport,
  ```

- To the BEGIN_COM_MAP statement:
  ```
  COM_INTERFACE_ENTRY(IVideoPTZPresetImport)
3.3.1 Obtain Camera Channel Count

HRESULT Lnl_GetCameraCount( int *count );

To determine how many cameras will be imported, the count must be obtained first. This function must return the exact number of cameras configured on the recorder whether they are online or not.

3.3.2 Retrieve the Camera IDs

HRESULT Lnl_GetCameraChannelIDs(  
    long pCameraIDs[], int *count );

This function obtains the full list of camera channel IDs with which the recorder is configured. These IDs are recorder-specific and are used to get or set the camera configurations whenever needed. On a full import, this function will be called followed by Lnl_GetCameraConfigByChannelID for each ID in the list returned to obtain all the cameras configured for the recorder. The count is populated with the size of the empty array but should be set to the number of array elements assigned valid values (as long as it is equal to or less than the size of the array). The caller of this function is the one responsible for cleaning up the memory allocated in pCameraIDs.

3.3.3 Retrieve a Camera Based on Channel ID

HRESULT Lnl_GetCameraConfigByChannelID(  
    long id, BYTE *pCameraData, DWORD cbSize );

While iterating through the cameras based on the count obtained in the previous function, this function retrieves the camera information for the camera with the ID retrieved within the list returned from the call Lnl_GetCameraIDs. This function may also be used to refresh OnGuard with any configuration changes that may have been made to the camera from outside OnGuard. The caller of this function is the one responsible for cleaning up the memory allocated in pCameraData.

pCameraData is passed as a BYTE array but is actually a CAMERA_DATA4 structure. Set cbSize to the size of CAMERA_DATA4 before the function is returned.

3.3.4 Retrieve the number of PTZ presets a camera has

HRESULT Lnl_GetCameraPTZPresetCount(  
    long cameraId, int *pCount );

In order to determine how many PTZ Presets will be imported for a camera, the count must be obtained first. This function must return the exact number of PTZ presets configured on the recorder for this camera.

3.3.5 Retrieve the PTZ presets information

HRESULT Lnl_GetCameraPTZPresets(  
    long cameraId, BYTE *pPTZPresets, long size);  

While iterating through the PTZ presets, based on the count obtained in the previous function, this function retrieves the preset information for the camera. The Preset information is stored in the pPTZPresets argument. The *pPTZPresets argument is an array of CAMERA_PTZ_PRESET.
CAMERA_PTZ_PRESET is defined as follows:

```c
typedef struct _CAMERA_PTZ_PRESET
{
    int PresetIndex;
    wchar_t PresetName[512];
} CAMERA_PTZ_PRESET;
```

### 3.4 Status and events

The status and event information for any OnGuard device displayed in Alarm Monitoring or acted upon by the Linkage Server comes from the Communication Server responsible for that particular device. The Communication Server (the workstation defined on the device’s configuration page in System Administration) always creates a single instance of the device translator that is responsible for reporting the status and events from that device. Alarm Monitoring (and other video clients, such as VideoViewer) creates its own instances of translators; however these instances are used for video playback only. The Communication Server and video client translator instances are initialized and used in a completely different way. This section describes the initialization and usage of the translator created within the Communication Server.

#### 3.4.1 Translator initialization

Status of the recorder and its channels is reported back to OnGuard via special “status” events. In addition to these events, translator can send out other pre-defined events and generic text events.

When the OnGuard Communication Server creates an instance of the translator for a particular recorder, it sets up a two way communication via connection point interfaces. The template project provides implementation of the relevant interfaces and stores the outgoing interface pointer to IDistributeEvent interface in a member variable. Typically, OAAP partners utilize the template project’s WriteEventToClients() function to send out events. This function takes one parameter, a pointer to LNLMESSAGE structure, which is described in the samples below. Most events need to be sent from the ITranslate::Lnl_PollPanelForEvents() function that the Communication Server repeatedly calls in order to avoid any multi-threading issues. Some events, such as the special status and firmware events described below, need to be called from the ITranslate::Lnl_GetSystemStatus(), function which is called repeatedly until a connection with the recorder is established. Special status events need to be sent from both Lnl_GetSystemStatus() and Lnl_PollPanelForEvents() functions.

```c
HRESULT Lnl_InitializePanel();
```

This method is called inside a loop to check if the recorder is still online, in which case S_OK needs to be returned. Typically this is the place where communication with the device (video recorder) is initialized.

```c
HRESULT Lnl_PollPanelForEvents();
```

This method is called from inside a loop to check if any events need to be reported. In order to make sure that events are always reported from the same thread (which is required), it is recommended that all events are reported from within this method. The only exception is the Lnl_GetSystemStatus() method where some events are also expected to be reported.

```c
HRESULT Lnl_GetSystemStatus(SYSTEM_STATUS* p_Status);
```

While this method provides an opportunity to return status synchronously via the p_Status parameter, it is recommended to send the status asynchronously (via special status events described below),
especially if obtaining the status takes a long time. The Communication Server creates a separate communication thread for a different device type, by default this thread will control up to eight devices of that type (this behavior can be changed via an \acs.ini\ file setting). In case more than one device is handled by the same thread, it is recommended that this function is implemented in such a way that it does not lock up the thread for a long time in order to provide users with better UI experience.

**HRESULT Lnl_SendPanelNextDirectCmd();**

This function is not needed for OpenVideo interfaced recorders. However, it is being called repeatedly from a loop, so to minimize CPU usage it is recommended that you put a Sleep(5) statement in the implementation and then return S_OK value.

### 3.4.2 Event synchronization

A unique event index can be stored as a cookie in OnGuard database. This is typically used in case of a catastrophic failure in the system, such as a Communication Server crash, etc. The event index that is generated by the third-party (not to be confused with the serial number that OnGuard generates via GetPanelEventSerialNumber() method) is then retrieved every time the Communication Server starts up and allows the translator to re-send those events that did not make it to OnGuard because of the failure.

There are the two methods used for the event synchronization:

**HRESULT Lnl_UpdatePanelEventIndex(long panelID, long eventIndex);**

This method belongs to the outgoing IDistributeEvent interface (the same one as used to send out events below). The pointer to this interface is passed to the translator during initialization and is stored as a member variable (m_pDistributeEvent) in the template project. Call this method every time you send an event to OnGuard. The panel id value (available via the GetPanelID() helper function described in the Configuration Parameters section above) should be passed via the first parameter and the unique event index via the second parameter.

**HRESULT Lnl_SetEventIndex(long vl_EventIndex);**

This ITranslate2 interface method is called by the Communication Server during translator initialization. Implementation of this method should store this event id and use it later (typically during the first call to Lnl_PollPanelForEvents()) to actually synchronize the events.

### 3.4.3 Status events

These special events control the appearance of the red and yellow X’s over the recorder and channel icons in the Alarm Monitoring and VideoViewer system trees. We expect most OpenVideo interfaced products to send these events to be consistent with how device status is reported for most other types of devices in OnGuard. These events need to be sent out in the ITranslate::Lnl_GetSystemStatus() function implementation (this function is called once initially after connection to the recorder has been established). Also, these events need to be sent out from the ITranslate::Lnl_PollPanelForEvents() function every time the status of the recorder or an individual channel changes.

An example follows showing how to send a status event for an individual channel, where nChannel and bOnline are assumed to be local variables that contain the channel number and online status, respectively.

To set the status of the recorder, simply leave the LNLMESSAGE.sb_DeviceID member at zero. LNLMESSAGE.sb_DeviceType needs to be set to zero as well.

```c
LNLMESSAGE ls_Event;
ZeroMemory( &ls_Event, sizeof(ls_Event));
ls_Event.sl_Size = sizeof(LNLMESSAGE);
```
ls_Event.sl_Time = (DWORD)g_oTimeConverter.GetCurrentGmtTime();
ls_Event.sl_SerialNumber = GetPanelEventSerialNumber();
ls_Event.ss_AccessPanelID = m_PanelID;
ls_Event.sb_DeviceID = nChannel;
ls_Event.sb_MessageType = LNLMSG_TYPE_STATUS;
ls_Event.sb_EventType = L_EVENTTYPE_SYSTEM;
ls_Event.sb_EventDataType = EVENT_DATA_TYPE_STATUSREQUEST;
ls_Event.su_EventData.us_StatusRequest.sl_StatusType = DATA_SRQ_COMM_STATE;
ls_Event.su_EventData.us_StatusRequest.sl_Status = bOnline ? 1 : 0;
ls_Event.sb_DeviceType = L_DEVICETYPE_CCTVCAMERA;
WriteEventsToClients(&ls_Event);

3.4.4 Firmware version events
These special events will display the firmware version of the recorder and/or individual cameras in the
Alarm Monitoring system tree next to the corresponding icons when the Options > Display Device
Firmware Revisions menu item is selected in Alarm Monitoring. These events should be sent out
once during implementation of the ITranslate::LnL_GetSystemStatus() function. The following is an
example of setting the firmware version of an individual channel, where nChannel indicates the
channel number; Minor and Major indicate the minor and major integer portions of the firmware
version. To set the firmware version of the recorder, simply leave the LNLMESSAGE.sb_DeviceID
member with a value of zero. LNLMESSAGE.sb_DeviceType needs to be set to zero as well.

LNLMESSAGE ls_Event;
ZeroMemory( &ls_Event, sizeof(ls_Event));
ls_Event.sl_Size = sizeof(LNLMESSAGE);
ls_Event.sl_Time = (DWORD)g_oTimeConverter.GetCurrentGmtTime();
ls_Event.sl_SerialNumber = GetPanelEventSerialNumber();
ls_Event.ss_AccessPanelID = m_PanelID;
ls_Event.sb_DeviceID = nChannel;
ls_Event.sb_MessageType = LNLMSG_TYPE_STATUS;
ls_Event.sb_EventType = L_EVENTTYPE_SYSTEM;
ls_Event.sb_EventDataType = EVENT_DATA_TYPE_STATUSREQUEST;
ls_Event.su_EventData.us_StatusRequest.sl_StatusType = DATA_SRQ_FIRMWARE_REV;
ls_Event.su_EventData.us_StatusRequest.sl_Status = MAKELONG(Minor, Major);
ls_Event.sb_DeviceType = L_DEVICETYPE_CCTVCAMERA;
WriteEventsToClients(&ls_Event);

3.4.5 Communication Lost/Restored events
These events are typically expected in addition to the status events described above in order for the
user to see the Communication Lost/Restored alarms in the Alarm Monitoring application. Unlike the
special status events, these events (as well as any other regular pre-defined or generic text events) can
be configured to trigger actions via Global I/O, as well as trigger other operations such as event
recording and automatic video launching. The following is an example of sending a Communication Lost event for an individual channel (for the recorder just leave the LNLMESSAGE.sb_DeviceID member with a value of zero and the LNLMESSAGE.sb_DeviceType needs to be set to 0 as well):

```c
LNLMESSAGE ls_Event;
ZeroMemory( &ls_Event, sizeof(ls_Event));
ls_Event.sl_Size = sizeof(LNLMESSAGE);
ls_Event.sl_Time = (DWORD)g_oTimeConverter.GetCurrentGmtTime();
ls_Event.sl_SerialNumber = GetPanelEventSerialNumber();
ls_Event.ss_AccessPanelID = m_PanelID;
ls_Event.sb_DeviceID = nChannel;
ls_Event.sb_DeviceType = L_DEVICETYPE_CCTVCAMERA;
WriteEventsToClients(&ls_Event);
```

### 3.4.6 Other predefined events

A set of pre-defined system and video events can be sent out in the same way as the Communication Lost/Restored events. Refer to the LNL_Events.h file that comes with the OnGuard SDK disc that contains definitions of all other pre-defined system and video events in OnGuard. Just set the LNLMESSAGE.sb_EventID member to any of the event that start with either L_SYSTEM_ or L_VIDEO_. For events that start with L_SYSTEM_, the LNLMESSAGE.sb_EventType member needs to be set to the L_EVENTTYPE_SYSTEM constant, while for events that start with L_VIDEO_, this member needs to be set to L_EVENTTYPE_VIDEO.

For example, to send out a motion detected event for a channel, just change the following two members of the LNLMESSAGE structure in the Communication Lost/Restored example above:

```c
ls_Event.sb_EventType = L_EVENTTYPE_VIDEO;
// use L_VIDEO_MOTION_DETECTED_RESTORED below for a restored event
ls_Event.sb_EventID = L_VIDEO_MOTION_DETECTED;
WriteEventsToClients(&ls_Event);
```

**Note:** For Motion Detected/Restored events the user must check the 'Display Motion Detection alarms' checkbox in the Camera tab (in System administration application). That is in order for the events not to get filtered (and therefore not displayed) by OnGuard. The user should select the check box for every channel that sends a motion detection alarm. The Motion Detected/Restored events have worked since early versions of OnGuard.

The Blind Camera events were get-filtered until OnGuard 5.12.110. In OnGuard 6.0.148 or later the events are enabled by default and are not get-filtered.

### 3.4.7 Generic text events

Generic text events can be sent when no match can be found in the list of predefined OnGuard events (or as a workaround, the Motion Detected and Blind Camera event filtering bug as described above). The following is an example of how to send a generic text event:
CString MessageText;
messageText = _T("Generic Text");
LNLMESSAGE ls_Event;
memset( &ls_Event, '0', sizeof(ls_Event));
ls_Event.sl_Size = sizeof(LNLMESSAGE);
ls_Event.sl_SerialNumber = GetPanelEventSerialNumber();
ls_Event.sl_Time = g_oTimeConverter.GetCurrentGmtTime();
ls_Event.ss_AccessPanelID = m_PanelID;
ls_Event.sb_DeviceID = nChannel;
ls_Event.sb_DeviceType = L_DEVICETYPE_CCTVCAMERA;
ls_Event.sb_EventType = L_EVENTTYPE_GENERIC;
ls_Event.sb_EventID = L_GENERIC_GENERIC_EVENT;
ls_Event.sb_EventDataType = EVENT_DATA_TYPE_STATUS;
ls_Event.sb_AssociatedText = 1;
ASSERT(!messageText.IsEmpty());
ASSERT(m_pDistributeEvent != NULL);
BSTR bstrMessageText = messageText.AllocSysString();
m_pDistributeEvent->Lnl_DistributeLnlMessageWithTextEx(&ls_Event, bstrMessageText);
::SysFreeString( bstrMessageText );

3.5 Video playback

Video clients such as Alarm Monitoring, VideoViewer, and Remote Monitor create separate and sometimes multiple translator instances for video playback. These instances are initialized and used differently from the single translator instance created within the Communication Server for event and status reporting. This section describes how the video client translator instances are initialized and used.

3.5.1 Translator initialization
The following methods are called during the initialization of the video client-created translator instances. Unless specified otherwise these methods belong to IVideo interface. Also, some of the methods such as Lnl_SetPlayerWindowPos() and Lnl_SetPlaybackSpeed() can be called any time after the initialization is complete.

HRESULT Lnl_InitializePlayer();

This method should return S_OK. It is called only once during translator creation and can be optionally used to allocate resources needed for future video playbacks. It is recommended that the implementation of the Lnl_CreatePlayerInstance() method checks to see if this method has been called and, if not, simply call it internally. This is to avoid potential problems when a client does not call this function. (This has been an issue before and might also become an issue when new clients are introduced.)

HRESULT Lnl_CreateCommunicationObject();

This method typically is not used for video integrations; the default implementation that simply returns
S_OK should be sufficient. There is at least one known instance where this method is not called during translator initialization, specifically when the Select PTZ action gets executed. If this method is used to perform critical initialization, it is recommended that safe guards similar to the one described for Lnl_InitializePlayer() method are in place to make sure this method is always called regardless of the client.

HRESULT Lnl_CreatePlayerInstance( HWND playbackWndHandle, HWND callbackWndHandle, long* playerInstance );

This method is called when the user attempts to playback live or recorded video. Translator needs to create a child window for the actual video playback and set the parent to the window passed in the playbackWndHandle parameter. The callbackWndHandle parameter is obsolete. This method needs to allocate and return a unique player instance number in the playerInstance parameter. This player instance number will be passed in the subsequent calls to Lnl_PlayLive() and Lnl_PlaySegment() method calls.

IMPORTANT: Typically OAAP partners create a proprietary ActiveX control responsible for video playback. In this case, it is responsibility of the OAAP partner to create the ActiveX container window that can host the ActiveX control. Both MFC and ATL libraries provide some helper classes (CWnd and CAxWindow respectively) that implement ActiveX container functionality. Here are some simple steps for using the MFC approach:

1. Create a CWnd object with parent handle set to playbackWndHandle.
2. Call CWnd::CreateControl() method on this object to add the video playback ActiveX control.

Note: When writing the COM code that blocks and waits for COM callbacks (i.e. event notifications), the code must use wait functions which continue to process COM calls. If this is not done, a deadlock situation is introduced.

HRESULT Lnl_ReleasePlayerInstance( long playerInstance );

This method is called just before the playback window is destroyed. This is a chance to release any resources that were allocated for the playback referred in the playerInstance parameter.

HRESULT Lnl_SetPlayerWindowPos( long playerInstance, RECT newWindowPos );

This method is called whenever video window is created or resized. The playerInstance parameter indicates player instance returned via a previous call to Lnl_CreatePlayerInstance() method. The newWindowPos parameter contains the new window location and size in client coordinates. No assumption should be made about initial size of the window, which is subject to change between different versions of OnGuard. It was changed specifically in 5.12.012 release; however this should not have any effect if this method is implemented correctly and video playback window that translator creates in Lnl_CreatePlayer() method gets resized properly.

HRESULT Lnl_SetPlaybackSpeed(long playerInstance, long playSpeed);

This method is called when the user changes the playback speed via the speed slider bar located in the recorded playback toolbar of the video window. The playerInstance parameter indicates the player instance returned via a previous call to Lnl_CreatePlayerInstance() method. The playSpeed parameter is a number from -5 to 5, where 0 is normal playback speed.
HRESULT GetVideoDimensions(long PlayerInstance, long* Width, long* Height);

This method is part of the IVideo2 interface and is called in order to determine the video resolution of the current playback. The video resolution is used to determine the initial video window size.

3.5.2 Live Video playback

HRESULT Lnl_PlayLive(long playerInstance, long CameraNumber, long playSpeed);

This method is called to play live video for the channel id in CameraNumber. playerInstance indicates the player instance that was previously returned by Lnl_CreatePlayerInstance() method. The playSpeed parameter is obsolete. Some methods, such as Lnl_SetPlayerWindowPos() only pass player instance id, so it is recommended that a map between the player instance id and information about the corresponding playback is maintained. This playback information typically contains references to the playback window and currently played channel, the time range of the recorded video (when applicable), etc. In some cases player instances are reused, which means that the Lnl_PlayLive() and Lnl_PlaySegment() methods can be called multiple times for the same player instance. Each call to these methods may refer to the same or a different channel and in the case of recorded video the time range can be changed on the fly as well. One thing that will not change through the lifetime of the translator instance is the recorder that it is responsible for.

3.5.3 Recorded Video playback

HRESULT Lnl_PlaySegment(long playerInstance, long CameraNumber, SYSTEMTIME StartTime, SYSTEMTIME EndTime, long playSpeed);

This method is called to play recorded video for the channel in CameraNumber. playerInstance indicates the player instance that was previously returned by Lnl_CreatePlayerInstance() method. playSpeed parameter is obsolete (the current recorded video playback speed is set via Lnl_SetPlaybackSpeed() method instead). The StartTime and EndTime indicate the time range for the requested recorded video segment. Note that this time is always in the local time zone associated with the recorder (this association is done in System Administration from the Digital Video > Video Recorder > Connection sub-tab via the World Time Zone drop-down list). The information about the time zone and DST setting can be retrieved via GetPanelTZInfo() and GetDST() helper functions provided by the translator template as described in the Configuration section above.

HRESULT Lnl_GetActualPlayTime(long playerInstance, SYSTEMTIME* StartTime, SYSTEMTIME* Endtime);

This method is called after each call to Lnl_PlaySegment() in order to get the actual start and end time that are available. This is a chance for the translator to adjust the time range requested in Lnl_PlaySegment in case video is not available for the requested start and/or end time.

3.5.4 Turn off Cell Hiding

When switching between live and recorded video in OnGuard, by default OnGuard keeps both streams connected. This is done for optimization purposes so when the end user switches between live and recorded video, the video will be displayed faster. This behavior can be disabled using the DVR capabilities file (see section 4). The <TurnOffCellHiding/> element needs to be added to the DVR capabilities file in order for OnGuard to close the live video stream after the end user switches to recorded video. This feature is available in OnGuard 6.5 or later.
3.5.5 Playback control

HRESULT Lnl_PlayStop(long playerInstance);
This method stops the recorded or imported video playback and resets the current playback position to
the beginning of the segment.

HRESULT Lnl_PlayPause(long playerInstance);
This method pauses recorded or imported video playback and also freezes the live video playback. The
current playback position should not be changed.

HRESULT Lnl_PlayResume(long playerInstance);
This method resumes previously stopped or paused recorded or imported video playback and unfreezes
live video playback.

HRESULT Lnl_PlayFastForward(long playerInstance);
This method fast forwards the current recorded or imported video playback.

HRESULT Lnl_PlayRewind(long playerInstance);
This method rewinds (plays backwards) the current recorded or imported video playback. Currently
this method is not utilized in OnGuard; there is no UI button that triggers calls to this method. Support
for this functionality may be added in the future.

HRESULT Lnl_PlaySeek(long playerInstance, long secFromStart);
This method is called when the user changes the current playback position by sliding the position
progress bar during recorded or imported video playback.

HRESULT GetCurrentPosition(long PlayerInstance, hyper* FiletimeRelativelyStartTime);
This method belongs to the IVideo2 interface and is called periodically to get the current playback
position. The returned value is used to update the position progress bar during recorded or imported
video playback. The FiletimeRelativelyStartTime parameter should be set to the current time relative
to the start of the currently playing video segment in nanoseconds.

HRESULT StepFrame(long PlayerInstance, long cFrames);
This method belongs to the IVideo2 interface. It is called to advance or reverse the number of frames
specified by the absolute value of the cFrames parameter. Positive values are used to advance that
many frames and negatives are used to go to a previous frame.

HRESULT CanStepFrame(long PlayerInstance, long Flags);
This method belongs to the IVideo2 interface. Return S_OK if advancing and/or reversing frames
functionality is supported via StepFrame. Otherwise, the corresponding UI control (frame advance
button) will be disabled.

Note: This method is no longer called by OnGuard for OAAP integrations.
3.6 Video Import and Export

The following methods of the IVideo interface are used for video import and export functionality. Import functionality is supported beginning with OnGuard 6.0. The Dynamic DVR Capabilities feature enables association between the video recorder and supported file extension. The Dynamic DVR Capabilities feature is based on an XML file provided by the OAAP partner that includes some capabilities of the OAAP recorder. One of the capabilities is the file extension(s) of the OAAP recorder. By indicating the file extension(s) in the XML file, OnGuard can associate it with the recorder type.

```csharp
HRESULT Lnl_ExportVideo( long playerInstance, long cameraNumber, SYSTEMTIME startTime, SYSTEMTIME endTime );
```

This method is called when the user selects the Options > Export Video menu option in the video window. playerInstance indicates the player instance that was previously created via Lnl_CreatePlayerInstance() method described in the Video Playback section above. cameraNumber indicates the channel, while startTime and endTime indicate the time range requested. Currently the requested video segment is always the same as the one being currently played back (set by the previous call to Lnl_PlaySegment()); however this may change in the future, so it is recommended that the values passed here are used for the export operation and not the ones stored in the player instance / playback information map mentioned in the Video Playback section. It is responsibility of this method’s implementation to prompt the user for the file name to be used, typically via a common Save As dialog. This dialog should be modal and its parent window should be set to the playback window associated with the player instance indicated in the playerInstance parameter. When possible, at least for files created in proprietary formats, it is recommended that unique file extensions are used, in order to avoid potential conflicts with other recorder types supported in OnGuard.

```csharp
HRESULT Lnl_ExportVideo2( long playerInstance, long cameraNumber, SYSTEMTIME startTime, SYSTEMTIME endTime, long cStrings, LV_STRING_DATA* pStrings);
```

A new VideoExport method was introduced in OnGuard 5.12.012. This method adds more information about the export operation. The method is part of the IVideo4 interface so this must be declared in the translator project. OnGuard still supports the ‘old’ method (Lnl_ExportVideo). When the user selects the Options > Export Video menu item, OnGuard checks if the IVideo4 interface is available in the translator object. If it is, the Lnl_ExportVideo2 is called. If not, the ‘old’ function is called. The first four parameters of the methods are the same as in the Lnl_ExportVideo method. The pString array includes the following information (in that order): 1) The time and date when the user export the video; 2) The camera name; 3) The OnGuard user name which was used in order to access the OnGuard application the ‘export’ was done from; 4) The recorder name. The cString variable indicates the size of the pString array.

```csharp
HRESULT Lnl_PlayFile( long playerInstance, BSTR fileName, long playSpeed );
```

This method will be called when video import functionality is enabled for OAAP recorders to playback video previously exported to a file via the Lnl_ExportVideo() method described above. The playerInstance parameter indicates the player instance returned via a previous call to Lnl_CreatePlayerInstance() method described in Video Playback section above. The fileName parameter contains the file name where exported video was stored. The playSpeed parameter is obsolete, currently the playback speed is set via calls to Lnl_SetPlaybackSpeed() method instead.

3.7 Change password

Supporting the change password feature allows the OnGuard user to change the recorder and/or camera
password. Changing the password is done from system administration application->Digital Video->Security tab.

3.7.1 Change recorder password
To allow changing the recorder password, the ILnrChangePassword interface needs to be implemented by the translator. The interface contains a single method.

The ILnrChangePassword interface definition can be obtained via include “Ln1.Video.Api\lnrapiu.h” statement.

The following is an example of how to add support for the ILnrChangePassword interface in the translator project.

Simply add the following lines to the VideoTrans.h file:

```csharp
#include “Ln1.Video.Api\lnrapi.h"
```

To the CVideoTrans declaration statement:

```csharp
public ILnrChangePassword,  
```

To the BEGIN_COM_MAP statement:

```csharp
COM_INTERFACE_ENTRY(ILnrChangePassword)  
```

**HRESULT ChangePassword(LPWSTR User, LPWSTR OldPassword, LPWSTR NewPassword)**

The User parameter should include the domain name in case the recorder is under a domain.

3.7.2 Change camera password
To allow changing the camera password, a separate ILnrRecorderChangePassword interface needs to be implemented. In addition, the ChangePasswordCapability camera capability element needs to be set for camera types that support password changes (refer to section 3.2.2.1). This element, besides indicating support for changing password, also describes the valid length and character set to be used for new password validation, which is done on OnGuard side. The ILnrRecorderChangePassword interface defines a single method:

**HRESULT ChangeChannelPassword(long ChannelID, LPWSTR User, LPWSTR OldPassword, LPWSTR NewPassword)**

Here is an example of the ChangePasswordCapability:

```xml
<SourceCapabilities>
...
  <ChangePasswordCapability>
    <MinLength>1</MinLength>
    <MaxLength>8</MaxLength>
    < CharSet>30-39,41-5A,61-7A</CharSet>
  </ChangePasswordCapability>
</SourceCapabilities>
```

**Note:** The number ranges in the CharSet element contain the ASCII character values allowed for the password.
3.8 Event recording

To implement the currently supported time-based event recording, the IVideoEventConfiguration interface must be implemented. The definition of this interface can be found in IVideo.idl file that comes with the OnGuard SDK disc.

IMPORTANT: The template GenericVideoTranslator project does not initially implement this interface. The CVideoTrans class needs to be derived from this interface and also an appropriate COM_INTERFACE_ENTRY statement needs to be added to the BEGIN_COM_MAP / END_COM_MAP section in order to expose this interface to OnGuard.

The IVideoEventConfiguration interface contains the following two methods:

HRESULT StartEventConfiguration( [in] long cameraNumber, [in] long dwTimeout);

HRESULT EndEventConfiguration( [in] long cameraNumber);

The cameraNumber parameter indicates the channel for which event recording mode needs to be turned on or off. The StartEventConfiguration() method is used to turn the event recording mode on and also takes an additional dwTimeout parameter which indicates the timeout value in seconds. Incase the EndEventConfiguration() method that turns the event recording mode off is not called within the timeout, the event recording mode should be turned off automatically.

3.9 Event locking

The following methods are called by the Communication Server when OnGuard attempts to lock or unlock a video event. Locking an event means that recorder should not delete associated video during any kind of recycling of the storage space. These methods are part of the published IVideo interface and default translator template project simply returns E_NOTIMPL value in their implementation. The following are the methods:

HRESULT Lnl_CreateEvent( long cameraNumber, SYSTEMTIME* pStartTime, SYSTEMTIME* pEndTime, long* pEventPercentageUsed );

This method is called in two cases: when an OnGuard alarm is configured to trigger event locking in System Administration via the Digital Video > Alarm-Video Configuration tab or when the user manually creates a user generated video event by selecting Options > Create Video Event menu item while playing back recorded video in a video window. The cameraNumber parameter indicates the channel id, pStartTime and pEndTime parameters contain the start time and end time of the event that needs to be locked. The pEventPercentageUsed parameter is currently not used. It may be used in the future to report the percentage of storage space that is occupied by locked video events in relation to total storage space configured to be available for the recorder.

HRESULT Lnl_DeleteEvent( long cameraNumber, SYSTEMTIME startTime, SYSTEMTIME endTime, long* pEventPercentageUsed );

This method is called when user right clicks on the event icon in the Alarm Monitoring alarm view and selects Unlock Event menu item. This menu item is only enabled for events that are currently locked (their icons are also marked blue in the alarm view to indicate locked mode). When support for OAAP recorders is added to OnGuard Archive Server, this method will also be called for each event that Archive Server successfully archives (when there is no need for the event to remain locked).

HRESULT Lnl_DeleteAllEvents();

Currently this method is called as part of trouble shooting process.
3.10 PTZ Control

OnGuard supports two types of PTZ control: server (optical) and client (digital) PTZ. While the server PTZ control can only be used while viewing live video, client (digital) PTZ can be used during recorded video playback as well.

There are also two types of server PTZ control supported in OnGuard: matrix switcher and recorder-based. No additional development is required to take advantage of the matrix switcher PTZ control, since all PTZ commands in this case are sent directly to the physical matrix switcher or to the PTZ dome camera (via virtual matrix switcher) typically using a serial connection.

Beginning with OnGuard 5.12.012, support has been added for the recorder-based server PTZ control to the OpenVideo API. This section of the document provides all the information needed to implement this new interface.

3.10.1 Windows messages

The parent window that OnGuard provides for video playback (the one which handle is passed to Lnl_CreatePlayer() method) is responsible for handling user input related to PTZ control. While some PTZ commands are provided via PTZ toolbar that gets created, others are issued via mouse clicks and key presses. It is common practice for OAAP partners to use their own ActiveX control to provide video playback functionality. It is also common for such controls to process many Windows messages, including mouse and keyboard messages. These messages need to be passed to the parent window that OnGuard provides so that they can be processed. Otherwise, OnGuard will not be able to issue PTZ commands in response to mouse and keyboard messages.

A list follows of the messages that are recommended for passing to the parent window:

WM_KEYDOWN, WM_KEYUP,
WM_MOUSEACTIVATE, WM_MOUSEMOVE, WM_NCMOUSEMOVE,
WM_LBUTTONDOWN, WM_LBUTTONUP,
WM_RBUTTONDOWN, WM_RBUTTONUP,
WM_MBUTTONDOWN, WM_MBUTTONUP,
WM_NCMBUTTONDOWN, WM_NCMBUTTONUP

3.10.2 Activation of the client (digital) PTZ control

In order to enable the Activate Digital PTZ button located on the video window toolbar in the Alarm Monitoring and VideoViewer applications, the following two IVideo2 interface methods must be implemented:

HRESULT SetSourceRectangle( long PlayerInstance, long Left, long Top, long Width, long Height );

This method is called when the user digitally zooms in or out. The PlayerInstance parameter refers to a previously created player instance via Lnl_CreatePlayerInstance() method. The Left, Top, Width, and Height parameters specify which area of the current frame needs to be visible and stretched to fill the whole playback area. These parameters are in pixels.

HRESULT GetSourceRectangle( long PlayerInstance, long* pLeft, long* pTop, long* pWidth, long* pHeight );
This method is called to read back the digital zoom area that was previously set via the SetSourceRectangle() method.

### 3.10.3 Activation of the server (optical) PTZ control

In order to enable the Activate Server PTZ button located on the video window toolbar in Alarm Monitoring and VideoViewer, the IVideo2::GetInternalObject() method needs to return a pointer to the new ILvPTZ interface when the first parameter (lObjectType) is equal to voServerPTZ constant (value 13). An example follows of the function implementation:

```cpp
STDMETHODIMP CVideoTrans::GetInternalObject(long lObjectType, long PlayerInstance, REFIID riid, void** pp)
{
    if (lObjectType != voServerPTZ)
        return E_NOTIMPL;
    InternalAddRef();
    *pp = (ILvPTZ*) this;
    return S_OK;
}
```

**IMPORTANT:** Beginning with the 5.12.110 release of OnGuard, the PlayerInstance parameter can have either positive or negative values. If the value is positive it should be treated as a player instance id that was previously returned by the Lnl_CreatePlayerInstance() method. However; if the value is negative, then its absolute value should be treated as the channel id. The channel id (as a negative number) will be passed to the translator when Linkage Server executes the Select PTZ Preset action (as part of Global I/O or Scheduler features) or when the new PTZ Tour Server runs a PTZ tour in the background (this server will be introduced in the next release of OnGuard). This is because both of these servers do not playback video and thus Lnl_CreatePlayerInstance() will never be called on the translators that these servers create. It is important that no windows are created when a negative player instance id (which means a channel id) is passed to the GetInternalObject() method, since these servers are typically running as services and do not have access to the user desktop.

The example above assumes that the ILvPTZ interface is implemented by the translator object itself. Alternatively a separate COM object could be designated for implementing the interface, in which case the IVideo2::GetInternalObject() function implementation would instead instantiate that separate object and query it for the ILvPTZ interface pointer to return. The example above assumes that the following lines were added to the VideoTrans.h file (in addition to the ILvPTZ interface method declarations):

**IMPORTANT:** When an instance of the device translator is created by the Linkage Server, the Pane ID variable is not initialized. However, the IP address of the panel does get initialized. The IP address should be used in order to find the panel (DVR) to which the channel id belongs.

To the declaration section:

```cpp
#include “Lnl.Video.Api\lnrapi.h”
```

To the CVideoTrans declaration statement:

```cpp
public IDispatchImpl<ILvPTZ, &__uuidof( ILvPTZ), &LIBID_LNRAPI >
```

To the BEGIN_COM_MAP statement:

```cpp
COM_INTERFACE_ENTRY(ILvPTZ)
```

To the .cpp file that has the main() function:

```cpp
#include “lnrapi_i.c”
```
It is recommended to save the value of the PlayerInstance parameter passed in the GetInternalObject() method above, since this value is used to determine which channel to control PTZ on. The PlayerInstance value is the one that Lnl_CreatePlayerInstance() method has previously returned for the video playback. Refer to the Video Playback section of this document for the description of PlayerInstance parameter. It is recommended to use a separate instance of an object that implements the ILvPTZ interface because in the future multiple simultaneously controlled PTZ channels may be supported. Currently, however, this is not a requirement, since only one channel can be PTZ controlled at a time.

3.10.4 PTZ Control interface
There are two PTZ modes available in OnGuard: continuous and relative (step). If the translator supports only one of these modes, OnGuard automatically uses the supported one (as specified in the GetPTZCapabilities() method described below). When both modes are supported, the 5.12.012 version of OnGuard uses relative (step) mode by default. Beginning with OnGuard 5.12.110, continuous mode is the default. Regardless of the version of OnGuard, the default behavior on the local machine can be overridden via acs.ini settings.

The following methods are part of the ILvPTZ interface:

**HRESULT GetPTZCapabilities(BSTR* pCapabilitiesXML);**

This method is called by OnGuard clients to enable or disable UI controls for specific PTZ commands based on the PTZ capabilities returned. The following is an example of this function’s implementation that returns all possible PTZ capabilities. Simply remove those lines that describe PTZ commands that are not supported by the channel. Note that the lines actually have to be removed, setting the value to 0 does not currently have any effect.

```c
STDMETHODIMP CVideoTrans::GetPTZCapabilities(BSTR* pCapabilitiesXML)
{
    if (!pCapabilitiesXML)
        return E_POINTER;
    CComBSTR bstrXML;
    bstrXML = _T("<PTZCapabilities>");
    bstrXML += _T("<PTZCapability name="SetContinuousPan">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetContinuousTilt">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetContinuousZoom">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetContinuousFocus">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetContinuousIris">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetRelativePan">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetRelativeTilt">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetRelativeZoom">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetAbsolutePan">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetAbsoluteTilt">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetAbsoluteZoom">1</PTZCapability>\n");  
    bstrXML += _T("<PTZCapability name="SetAbsoluteIris">1</PTZCapability>\n");  
    pCapabilitiesXML = bstrXML;
}
```
bstrXML += _T("<PTZCapability name="SetAbsoluteFocus">");
bstrXML += _T("1</PTZCapability>");
bstrXML += _T("<PTZCapability name="SetAbsoluteIris">");
bstrXML += _T("1</PTZCapability>");
bstrXML += _T("<PTZCapability name="GetAbsolutePan">");
bstrXML += _T("1</PTZCapability>");
bstrXML += _T("<PTZCapability name="GetAbsoluteTilt">");
bstrXML += _T("1</PTZCapability>");
bstrXML += _T("<PTZCapability name="GetAbsoluteZoom">");
bstrXML += _T("1</PTZCapability>");
bstrXML += _T("<PTZCapability name="GetAbsoluteFocus">");
bstrXML += _T("1</PTZCapability>");
bstrXML += _T("<PTZCapability name="GetAbsoluteIris">");
bstrXML += _T("1</PTZCapability>");
bstrXML += _T("<PTZCapability name="PTZLocking">");
bstrXML += _T("1</PTZCapability>");
*pCapabilitiesXML = bstrXML.Detach();
return S_OK;
}

HRESULT Move(int Pan, int Tilt, int Zoom, int Focus, int Iris, int UserID, int Priority, int TimeoutSecs, VARIANT_BOOL* AccessAllowed);

This method is called when continuous PTZ mode is on and the user clicks anywhere inside the video window with the left mouse button or clicks on the zoom, focus, or iris buttons. When such button is released, the same method will be called with the pan, tilt, zoom, focus, and iris parameters set to 0.

The pan, tilt, zoom, focus, and iris parameters range in values from -100 to 100. The absolute values indicate the speed with 1 being the lowest and 100 being the highest speed. Negative values represent pan left, tilt down, zoom out, focus far, and iris close commands, while positive values represent pan right, tilt up, zoom in, focus near, and iris open commands.

For a description of the UserID, Priority, TimeoutSecs and AccessAllowed parameters, refer to the description of the CheckPriority() function below. Generally, implementations of the Move(), MoveBy(), MoveTo(), and SelectPreset() methods should internally call CheckPriority() method to set the value of AccessAllowed.

Here is an example:

HRESULT CVideoTrans::Move(int Pan, int Tilt, int Zoom, int Focus, int Iris, int UserID, int Priority, int TimeoutSecs, VARIANT_BOOL* AccessAllowed);
{
    HRESULT hr = CheckPriority(UserID, Priority, TimeoutSecs, AccessAllowed);
    if (hr != S_OK || (*AccessAllowed) != VARIANT_TRUE)
    {
        return hr;
    }

    // go on and execute the ptz command here
    return S_OK;
}

HRESULT MoveBy(float Pan, float Tilt, float Zoom, float Focus, float Iris, int UserID, int Priority, int TimeoutSecs, VARIANT_BOOL* AccessAllowed);

This method is called when relative PTZ mode is on and the user double clicks or drags the image inside the video window or clicks and releases the zoom, focus, or iris button.
The pan and tilt parameters range in values from -100 to 100, where left top corner corresponds to the pan value -100 and tilt value 100. The point passed via pan and tilt parameters should move to the center of the video window, which is what called “click to center” functionality. Dragging the image (versus double clicking) will result in multiple “click to center” calls to this method.

The Zoom parameter indicates the zoom factor. Values between 0 and 1 indicate zoom out command, while values greater than 1 indicate zoom in command. For example, the value 2 indicates 2x zoom in command, while the value 0.5 indicates a 2x zoom out command (1/0.5).

The focus and iris parameters range in values from -100 to 100. Negative values indicate focus far and iris close commands, while positive values indicate focus near and iris open commands. By default, values -2 and 2 are always sent for these parameters. It is up to the OAAP partner to decide how much of a movement should correspond to these values. The absolute values of these parameters can be adjusted in OnGuard acs.ini file settings (they will only apply on the local machine). The following is an example of how to set the default absolute values to 50 for both focus and iris commands:

```
[DigitalVideo]
RelativeFocusStep=50
RelativeIrisStep=50
```

**HRESULTGetPosition(float* Pan, float* Tilt, float* Zoom, float* Focus, float* Iris);**

This method is called when the user creates a client side preset and should return the current absolute position of the camera. These values will then be stored in OnGuard database under a user provided name.

**HRESULT MoveTo(float Pan, float Tilt, float Zoom, float Focus, float Iris, int UserID, int Priority, int TimeoutSecs, VARIANT_BOOL* AccessAllowed);**

When the user selects a client side preset (created with the help of GetPosition() method) from the preset drop-down list in Alarm Monitoring or VideoViewer video window, this method is called with the pan, tilt, zoom, focus, and iris parameters set to the values that were previously returned via GetPosition() method and stored under the client side preset’s name in OnGuard database.

**HRESULT SelectPreset(int Preset, int UserID, int Priority, int TimeoutSecs, VARIANT_BOOL* AccessAllowed);**

This method is called to trigger a camera side preset. It is called when the user types in a number into the edit portion of the preset combo box and clicks the Go To Preset button located next to the combo box. The Preset parameter indicates the preset number typed in by the user.

**HRESULT CreatePreset(int Preset, int *pPreset)**

This method is for future use only; currently it is never called in OnGuard. In a future version of OnGuard it will be used to create a camera side preset. Currently, camera side presets must be managed (created, removed or modified) using third-party tools, for example IP based camera’s web page, etc.

**HRESULT RemovePreset(int Preset);**

This method is for future use only; currently it is never called in OnGuard. In a future version of OnGuard it will be used to remove a camera side preset.
HRESULT CheckPriority(int UserID, int Priority, int TimeoutSecs, VARIANT_BOOL* AccessAllowed);

This function is called when user attempts to activate Server PTZ on a channel and should check if the user specified in UserID has access to PTZ at this time. When <PTZLocking> element is specified in the PTZ capabilities xml string that is returned in the GetPTZCapabilities() method, the OnGuard application will deny access to PTZ when this method returns access denied. Originally the return value of S_FALSE indicated access denied; however the AccessAllowed parameter was added to support clients that are script based (where they can not access the actual returned HRESULT value). OnGuard VideoViewer (browser-based client) is an example of such client (it does not support OAAP recorders yet, but it will in a future release of OnGuard). Thus some clients today rely on the return value, while others rely on AccessAllowed value (or both), which means that both of these values need to be set appropriately. The return value should be S_OK and the AccessAllowed value needs to be set to VARIANT_TRUE when access is allowed. The return value should be S_FALSE and AccessAllowed value needs to be set to VARIANT_FALSE in order to deny access.

The following is an example of how the CheckPriority() method can be implemented. First, the following member variable declarations are used in the example below:

```cpp
int m_UserID;  // stores the id of the user that currently owns PTZ control
int m_UserPriority;  // stores the priority of the current user
COleDateTime m_Expires;  // stores the time when the PTZ lock expires for the current user
bool m_bPTZIsLocked;  // indicates if PTZ control is currently locked
```

These parameters need to be initialized in the constructor as follows:

```cpp
m_UserID = -1;
m_UserPriority = -1;
m_Expires = COleDateTime::GetCurrentTime();
m_bPTZIsLocked = false;
```

Here is the CheckPriority() method implementation:

```cpp
HRESULT CVideoTrans::CheckPriority(int UserID, int Priority, int TimeoutSecs, VARIANT_BOOL* AccessAllowed)
{
    *AccessAllowed = VARIANT_TRUE;
    if (!m_bPTZIsLocked)
    {
        m_UserID = UserID;
        m_UserPriority = Priority;
        if (TimeoutSecs > 0)
        {
            m_bPTZIsLocked = true;
            m_Expires = COleDateTime::GetCurrentTime();
            COleDateTimeSpan dtSpan(0, 0, 0, TimeoutSecs);
            m_Expires += dtSpan;
        }
    }
    return S_OK;
}
```

if (m_UserID != UserID)
{
    if (Priority <= m_UserPriority)
    {
        *AccessAllowed = VARIANT_FALSE;
    }
}
if (m_Expires.GetStatus() == COleDateTime::valid)
{
    COleDateTime dtCurrent = COleDateTime::GetCurrentTime();
    if (dtCurrent <= m_Expires)
    {
        *AccessAllowed = VARIANT_FALSE;
        return S_FALSE;
    }
}
else
{
    *AccessAllowed = VARIANT_FALSE;
    return S_FALSE;
}
}
else if (TimeoutSecs < 1)
{
    m_bPTZIsLocked = false;
    return S_OK;
}
if (TimeoutSecs < 1)
    return S_OK;

m_UserID = UserID;
m_UserPriority = Priority;
m_Expires = COleDateTime::GetCurrentTime();
COleDateTimeSpan dtSpan(0, 0, 0, TimeoutSecs);
m_Expires += dtSpan;
return S_OK;

3.11 Camera online/offline

The ‘Online’ checkbox in the camera page is now enabled for OAAP partners. It needs to be set in the DVR Capabilities XML file in order to appear in the OnGuard UI and used.

The ‘Online’ parameter is part of the CAMERA_DATA structure that is send using the Lnl_SetCamera() method (refer to section 3.1.1).

3.12 Detect Recorder Type and Cameras

3.12.1 Detect recorder type

Detection of recorder type is available for OAAP partners starting OnGuard 6.1. The feature is not part of the DVR Capabilities XML file.

HRESULT DetectRecorderType(BSTR *pType);

In order to support the ‘Detect’ feature, the DetectRecorderType() method needs to be implemented in the translator. The method is part of the IVideoEnum interface. The pType parameter should return the recorder name.
The method can be implemented as follow:

```c
STDMETHODIMP CVideoTrans::DetectRecorderType(BSTR *pType)
{
    *pType = ::SysAllocString(L"OAAPTest");
    return NOERROR;
}
```

### 3.12.2 Camera Wizard

The camera wizard feature is available for OAAP partners starting OnGuard 6.1. The feature needs to be enabled in the Capabilities file.

**HRESULT DetectChannels( long cChannelsSize, long *pChannels, long *pChannelsLen);**

In order to support ‘Camera Wizard’, the DetectChannels() method needs to be implemented in the translator. The method is part of the IVideoEnum interface.

The cChannelsSize is the maximum number of channels the camera wizard can support. If the recorder includes more channels than this number, an error should be return. The pChannelsLen parameter should indicate how many channels are actually available on the recorder. The pChannels parameter can now be ignored. It is recommended to put there zero memory, as shown below.

```c
::ZeroMemory(pChannels, *pChannelsLen*sizeof(long));
```

### 3.13 Recorder Configuration

**HRESULT Lnl_SetRecorder( BYTE* pRecorderData, DWORD cbSize);**

The Lnl_SetRecorder method is part of the IVideoRecordConfig interface. The only other interface method that needs to be implemented is Lnl_SetCamera. The Lnl_SetCamera method will be called instead of the Lnl_SetCamera method that is part of the IVideo interface.

The pRecorderData parameter is from the VIDEO_RECORDER_DATA2 structure type.

The features that can be set from OnGuard using the Lnl_SetRecorder method are:

- Delete old video files – ‘AutoDeleteDays’ parameter.
- Send alarm if unable to keep video on the recorder – ‘vrd2DaysKeepVideo’ parameter.
- Video resolution – ‘VideoSize’ parameter.
- Video standard – ‘VideoStandard’ parameter.

The features should be set in the DVR Capabilities XML file in order to appear in the OnGuard UI and used. The ‘Delete old video files’ and ‘Send alarm if unable to keep video on the recorder’ features are enabled together in the UI (with the <DeleteOldVideo> element).

#### 3.13.1 Send alarm if unable to keep video on the recorder

The following alarms need to be sent as part of the ‘Send alarm if unable to keep video on the recorder’ feature:

1.  EvenID = L_RECORDER_STORAGE_SHORTAGE; EventType = L_EVENTTYPE_VIDEO.
The alarm indicates that video storage will be full in less number of days than defined by the user.

2. EventID = L_RECORDER_STORAGE_SHORTAGE_RESTORED; EventType = L_EVENTTYPE_VIDEO. The alarm indicates that there is no space available to record to, so the oldest file will be deleted in order to continue recording.

3. EventID = L_RECORDER_STORAGE_FILE_DELETED; EventType = L_EVENTTYPE_VIDEO. The alarm indicates that the recording rate drops to allow the video storage to be full in the same or more number of days than defined by the user.

3.14 Archive Video

The following methods are part of the IVideo interface and used in order to support the Lenel archive video application. All the methods described below are also mentioned in the OpenDevice-InterfacesGuide.pdf file.

The archive video application also needs to be enabled from the DVR Capabilities file. Note that not only the <Archive> needs to be included in the XML file, but also the 'FileExtension' attribute with the file extension that needs to be archived.

```
HRESULT Lnl_InitializeArchiver();
```

The purpose of this function is to allow the translator to initialize anything that is required for archiving. This function will be called after it was checked that there is a connection to the video recorder. After the initializing part, the function should call the ITranslate:: Lnl_SetPanelState with a parameter of ITranslate:: PANEL_STATE_READY.

The method can be implemented as follow:
```
HRESULT Lnl_InitializeArchiver()
{
    //initialize anything require for archiving

    //Set the panel state to 'online'
    SetPanelState(PANEL_STATE_READY);
    return NOERROR;
}
```

```
HRESULT Lnl_DownloadVideo(long cameraNumber, SYSTEMTIME* pStartTime, SYSTEMTIME* pEndTime, BSTR fileName, long* pStopHandle);
```

The purpose of this function is to request video from the Video Recorder in order to download it to the archiving location. In other words, the function should write the video file to the archive location. The cameraNumber parameter indicates the camera ID number. The pStartTime and pEndTime parameters are the start and end time of the requested video. The times are in the video recorder local time. The fileName parameter is the path, in UNC format (\ServerName\DirectoryName\FileName.ext), to where the video recorder should write the video file to. The path includes the video file name. The pStopHandle is the handle the user can use in order to stop the video download.

The pStartTime and pEndTime parameters should be set as output parameters with the actual start and end time of the video file that downloaded to the archive location.

```
HRESULT Lnl_DownloadVideoProgress(long stopHandle, HRESULT* pStatus, hyper* pCurrentPos);
```

The purpose of this function is to check the progress of a download video request. This function will be called
repeatedly by the Archive Server to get the status of the download.

The stopHandle is the handle that was created on the Lnl.DownloadVideo call. The paStatus parameter can have the value ‘STILL_ACTIVE’ to indicate that the video is still being downloaded. The value ‘1’ indicates the video was downloaded successfully. The pCurrentPos parameter indicates what percentage of the video was already downloaded.

HRESULT Lnl_CancelDownloadVideo(long stopHandle);

The function should stop exporting the video to the archive server. The stopHandle parameter is the handle that was created on the Lnl.DownloadVideo call.

HRESULT Lnl_GetNextVideoBlock( long cameraNumber, SYSTEMTIME startTime, SYSTEMTIME* pStartTime, SYSTEMTIME* pEndTime );

The function is called in order to get the start time (pStartTime) and end time (pEndTime) of the video block on the video recorder that is the nearest to the startTime parameter. The cameraNumber parameter represents the channel number. The pStartTime and pEndTime parameters are later used (if there are no gaps/deleted video) in the Lnl.DownloadVideo function.

### 3.14.1 Event archiving

The following methods need to be implemented in order to support event archiving. With event archiving, the archiving server will archive and purge (or just purge) the locked events in the video recorder. More about locked events can be found on section 3.9.

HRESULT Lnl_SetVideoEventsThreshold( long eventThreshold );

The function is called in order to set the event archiving threshold in the video recorder. The threshold is the percentage of the recorder’s disk space allowed for locked video.

The video recorder should check the threshold against the actual space in the recorder’s disk, when a video event is locked. Once the threshold is reached an event should be sent to Alarm Monitoring. The event id is L_VIDEO_EVENTTHRESHOLD_REACHED (the event type is L_EVENTTYPE_VIDEO). Every 1% change above the threshold, the alarm should be sent.

Once the threshold is exceeded with 5%, an event with event id of L_VIDEO_DISK_FULL should be sent to Alarm Monitoring (the event type is L_EVENTTYPE_VIDEO).

HRESULT Lnl_GetVideoDiskInfo( long* pEventPercentageUsed );

The function should return the percentage of the video recorder’s disk space that is used by locked video events. The pEventPercentageUsed parameter should set with the percentage value. The function is called by the archive server in order to check if the threshold that was set for event archiving has been reached.

HRESULT Lnl_DeleteEvent( long cameraNumber, SYSTEMTIME startTime, SYSTEMTIME endTime, long* pEventPercentageUsed );

The function is described in section 3.9. The archive server calls this method in order to unlock (purge) the locked events. This method must be implemented when archive server is used.

### 3.14.2 Continuous archiving

The following method needs to be implemented in order to support a specific feature in the continuous archiving. With continuous archiving the archive server will continuously archive video from all the channels, which are configured to be archived.
Continuous archiving is configured per channel. The channel configuration is set using the Lnl_SetCamera() method (refer to section 3.1.1). The CAMERA_DATA structure includes the ‘ContArchive’ and ‘ContArchiveTimeZone’ parameters. These parameters store the archiving configuration of the channel. The ‘ContArchive’ parameter indicates if continuous archiving is supported for the channel or not. The ‘ContArchiveTimeZone’ parameter indicates the time zone the archiving should take place.

**HRESULT GetVideoAmount( long Flags, hyper Time, hyper* Size, hyper* TotalSize);**

The method is part of the IVideo2 interface. The method is called in order to see if the continuous archiving threshold has been met. The continuous archiving threshold is configured using the ‘Delay archiving until video buffer is less than’ checkbox.

The Flags parameter should have the value ‘0’ in order to execute the function. The TotalSize parameter should return the total amount of recorded video in the video recorder. The Size parameter should return the amount of recorded video, which is more recent than the Time parameter. The TotalSize and Size parameters should be returned with the same size unit (Mbytes, for example).

3.14.2.1 Delay archiving feature

The ‘Delay archiving until video buffer is less than’ checkbox allows to reduce the duplication of the video between the Recorder and the Archive Server. For example, the total capacity of the recorder and the archive server are 50 days of recording. In order to get total capacity of 90 days of recorded video in the system, the system is configured not to copy the most recent video to the archive server. Instead, copy only video from the Recorder that is older than 40 days old. Therefore, the most recent 40 days of video will be on the Recorder. The next 50 days will be on the archive server. The days 40-50 will exist in both places.

The number of days need to be converted into percentage in order configure the threshold. In the example total recorder capacity is 50 days, but the recent 40 should not be copied. Therefore the threshold should be set to 20%.

3.15 Set Clock

The ‘Set Clock’ feature enables time synchronization between the OnGuard system and the OAAP recorder. The Lnl_SetClock() function need to be implemented in order to support the feature. ‘SetClock’ needs to be enabled from the capability file. If it is not specified in the capability file, no call is going to be made to the Lnl_SetClock() function. Including the <SetClock> element in the capability file will activate automatic clock synchronization between OnGuard to the OAAP recorder.

When the SetClock feature is enabled from the capabilities file, a new checkbox - Automatic Clock Synchronization - appears on the Recorder tab in System Administration application. The checkbox allows the user to disable the automatic calls of Lnl_SetClock. The user can manually call setClock from the OAAP Recorder context menu in System Administration and Alarm Monitoring application.

**HRESULT Lnl_SetClock (long vl_Time, BOOL vb_WaitForRsp );**

The method is part of the ITranslate interface. The vl_Time parameter is the time that needs to be set on the recorder. The vb_WaitForRsp parameter indicates whether to wait for response (TRUE) or not (FALSE). The time passed into the Set Clock function (vl_Time parameter) is in UTC. The time passed in can be converted from UTC to the video server’s local time before setting the video server’s clock. The video server local time zone is available from the GetPanelTZInfo() function (for more information, refer to section 3.1.2).
3.16 Dry Contacts (Inputs and Outputs)

Recorder’s inputs and outputs can be integrated to the OnGuard system. The interface that needs to be implemented in order to integrate Recorder’s inputs is IInput. For Recorder’s outputs integration, the IOutput interface needs to be implemented. Both IInput and IOutput interfaces are documented on the OpenDevice-InterfacesGuide.pdf file.

Note that the GenericVideoTranslator template project does not implement the IInput and IOutput interfaces. The CVideoTrans class will need to be derived from these interfaces. Additionally, the COM map will need be updated with new entries (CON_INTERFACE_ENTRY) regarding the IInput and IOutput interfaces. That is in order to expose them to OnGuard.

The Dry Contacts feature needs to be enabled from the DVR Capabilities file. Note that there are separate element for inputs and outputs: The <DryContactsInput> element is for inputs support and the <DryContactsOutput> is for output support.

3.16.1 Inputs/Outputs configuration in OnGuard

The configuration of recorder inputs and outputs is done through the Alarm Panels folder in the System:Administration application (Access Control-> Access Panels). When an alarm panel is added, the OAAP recorder will show up in the Panel combo box (that is only when the feature is enables from the DVR Capabilities file). The panel types that will show up in the Type combo box when the OAAP recorder panel is chosen are: “Recorder Dry Contacts” and “Recorder Output Panel.”

Note: The types that are displayed are based on what is defined in the DVR Capabilities file. For example, if only the <DryContactsInput> element is defined, then only “Recorder Dry Contacts” will be displayed.

The “Recorder Dry Contacts” type serves as an input panel. The Alarm Input tab is where recorder inputs can be configured. The “Recorder Dry Contacts” serves as output panel. Using this panel type enables the user to configure the recorder outputs in the Alarm Outputs tab.
Only one (1) input panel can be assigned to the OAAP recorder. The panel can have up to 16 inputs. The OAAP recorder can be configured with only one (1) output panel. The panel can have up to 16 outputs.

3.17 **Failover and redundancy**

3.17.1 **Overview**

Failover functionality in OnGuard allows users to configure an additional recorder to record video from a channel when that channel’s primary recorder is down. This additional recorder is sometimes referred to as failover or secondary recorder. Redundancy allows users to configure the secondary recorder to record video all the time regardless of the primary recorder’s status. In order to enable configuration UI for your recorder type, recorder capabilities xml needs to contain `<Failover/>` and `<Redundant/>` elements.

3.17.1.1 **Configuration**

1. Launch the System Administration application.
2. Go to the Video / Digital Video / Camera tab.
3. When adding or modifying a channel, select a valid recorder from the Failover Recorder combo box (just beneath the Recorder combo box) and, optionally, select the Enable Redundancy check box.

3.17.1.2 **Default configuration**

Optionally, you can setup per-recorder default settings for the failover recorder and redundancy channel settings that will be automatically used whenever channel is added. This can be done via Video / Digital Video / Video Recorder / Failover tab in System Administration. These settings can always be changed on per channel basis during add operation or later via modify operation.

3.17.1.3 **Automatic offline configuration**

This feature is not directly related to failover and redundancy support, however it is typically used in conjunction with failover and redundancy in order to automatically mark a recorder logically offline if it physically went down and has not come back up within specified timeout. When recorder is marked logically offline (manually or automatically), no communication attempts will be made with that recorder in order to get status, events or video until it is manually marked back online. This can be done either from System Administration / Video / Digital Video / Video Recorder tab or by right clicking on that recorder in Alarm Monitoring hardware tree and selecting Mark Online. The automatic offline timeout can be specified on System Administration / Video / Digital Video / Video Recorder / Failover tab.

3.17.1.4 **Redundant channel events pass through**

By default, redundant channel events, which mean channel level (versus recorder level) events coming from redundant recorder while primary recorder is physically online, are being thrown out by OnGuard.
Communication Server. This is to avoid duplicate channel events, such as motion detection, etc… This behavior does not apply to status events for the channel, which are used to show both primary and secondary recorder connection status for redundant channels in Alarm Monitoring hardware tree via special icons. This default behavior can however be overwritten system wide by adding the following entry into the LNLCONFIG table in OnGuard database:

<table>
<thead>
<tr>
<th>LNLVALUE</th>
<th>LNLSTRING</th>
<th>LNLCONFIGID</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td>161</td>
<td>NULL</td>
</tr>
</tbody>
</table>

3.17.2  Additional implementation steps

3.17.2.1  Overview

Below are all the implementation changes (some required and some optional) that need to be implemented in order to add failover and redundancy support to the translator for your recorder type.

3.17.2.2  IVideo5 (IVideo.idl)

This interface is required for failover and redundancy support. Translator needs to implement this interface in order to receive failover recorder and channel information during video playback. This interface contains a single method:

```
HRESULT Lnl_SetPlayerViewSite(long playerInstance, IUnknown* pSite);
```

This method is called after IVideo::Lnl_CreatePlayerInstance() method and before any calls to IVideo::Lnl_PlayLive() or IVideo::Lnl_PlaySegment() methods. playerInstance parameter represents current playback instance id (see Lnl_CreatePlayerInstance documentation for more details). pSite parameter should be used to query ILvMultiSourceSite interface described later in this section.

3.17.2.3  IVideo4 (IVideo.idl)

This interface is optional. However, it needs to be implemented whenever IVideo5 is needed because IVideo5 derives from IVideo4. The single method of this interface is Lnl_ExportVideo2(). Note that OnGuard will call this method instead of the original IVideo::Lnl_Export() method if IVideo4 interface is supported, so in order not to break existing video export functionality Lnl_ExportVideo2() method implementation should simply delegate to IVideo::Lnl_ExportVideo method unless additional parameters described in section 3.6 are needed. An example follows of the Lnl_ExportVideo2() method implementation:

```
STDMETHODIMP CVideoTrans::Lnl_ExportVideo2(long playerInstance, long cameraNumber, SYSTEMTIME startTime, SYSTEMTIME endTime, long cStrings, LV_STRING_DATA* pStrings)
{
    return Lnl_ExportVideo(playerInstance, cameraNumber, startTime, endTime);
}
```

3.17.2.4  IVideo3 (IVideo.idl)

This interface also needs to be implemented because IVideo4 is derived from it. All methods of this interface should simply return E_NOTIMPL;

3.17.2.5  ILvMultiSourceSite (lnlView.idl)

OnGuard provides implementation of this interface by passing its pointer via
IVideo5::Lnl_SetPlayerViewSite() method described earlier in this section. Translator does not need to implement this interface - instead it needs to use it to get failover information. Here are the descriptions of the ILvMultiSourceSite methods:

HRESULT GetSourceCount(DWORD* pdwCount);

This method returns the number of sources configured in OnGuard for the channel associated with the current playback instance.

HRESULT GetSourceInfo(DWORD dwSrcIdx, BSTR* pdwIPAddress, LONG* pChannel, BSTR* pUsername, BSTR* pPassword, BOOL* pbEnabled, DWORD* pdwSourceType);

Once you get the number of sources from GetSourceCount, you can get specific source information by calling the GetSourceInfo method. Here are the parameter descriptions for this method:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwSrcIdx</td>
<td>0-based source index.</td>
</tr>
<tr>
<td>pdwIPAddress</td>
<td>IP address of the recorder.</td>
</tr>
<tr>
<td>pChannel</td>
<td>Channel number specific to the source. For primary recorders, this is the</td>
</tr>
<tr>
<td></td>
<td>number assigned on the Camera tab in System Administration. For failover</td>
</tr>
<tr>
<td></td>
<td>recorders, this number is automatically assigned to the next available</td>
</tr>
<tr>
<td></td>
<td>number when the channel is configured for failover in System Administration.</td>
</tr>
<tr>
<td>pUsername, pPassword</td>
<td>User credentials for the recorder (empty if security is not</td>
</tr>
<tr>
<td></td>
<td>supported by the translator or if nothing is configured for these on the</td>
</tr>
<tr>
<td></td>
<td>Recorder tab in System Administration).</td>
</tr>
<tr>
<td>pbEnabled</td>
<td>This flag indicates whether or not this recorder is currently marked</td>
</tr>
<tr>
<td></td>
<td>logically online or offline. It is important to avoid any communication</td>
</tr>
<tr>
<td></td>
<td>attempts with the recorder that is marked logically offline in order to</td>
</tr>
<tr>
<td></td>
<td>avoid any delays and also to avoid causing interference issues in case the</td>
</tr>
<tr>
<td></td>
<td>recorder is undergoing maintenance.</td>
</tr>
<tr>
<td>pdwSourceType</td>
<td>Possible values are 1 for the primary recorder and 2 for the secondary</td>
</tr>
<tr>
<td></td>
<td>recorder. These values correspond to stPrimaryRecorder and stSecondary</td>
</tr>
<tr>
<td></td>
<td>Recorder values for the LvSourceType enumeration type defined in lnlView.idl.</td>
</tr>
</tbody>
</table>

3.17.2.6  ILvMultiSource (lnlView.idl)

This interface is optional. It needs to be implemented for each playback instance and returned to the client via IVideo2::GetInternalObject() method implementation in order to receive notifications from OnGuard whenever a recorder is marked logically online or offline (manually or automatically) and to provide OnGuard with the currently active source information for the current playback. Here are the steps required to add support for these features:

IVideo2::GetInternalObject() method

This method is described in section 3.10.3 as part of functionality required to support recorder based PTZ control. For the failover features above, support for new object type, voMultiSource, needs to be added to the existing implementation of this function. Here is an example that handles both PTZ and failover object retrieval:

STDMETHODIMP CVideoTrans::GetInternalObject(long lObjectType, long PlayerInstance, REFIID riid, void** pp)
{
    if (lObjectType == voServerPTZ)
    {
        // place existing ptz logic here
    }
    else if (lObjectType == voMultiSource)
    {
        CComObject<CMyMultiSourceClass> ptrMultiSource;
        ptrMultiSource.CreateInstance();
        ptrMultiSource ->Init(this, PlayerInstance);
        return ptrMultiSource ->QueryInterface(riid, pp);
    }
    return E_NOTIMPL;
}

The following assumptions were made in the sample code above: CMyMultiSourceClass is your custom C++ class that is derived from ATL’s CComObjectRootEx template class and implements ILvMultiSource interface described below. Certainly it is not required to use ATL in this case – it is used here only in order to keep the sample concise. It also has initialization routine Init() that in this example takes pointer to the translator class and playback instance id, but could take anything that allows CMyMultiSourceClass to later switch sources and/or detect the currently active source in response to ILvMultiSource method calls from OnGuard. Note also that the sample above lacks any exception handling for brevity purposes.

HRESULT GetActiveSourceIndex(REFERENCE_TIME rtPos, DWORD* pdwSrcIdx);

This ILvMultiSource method is used by OnGuard to show or hide secondary recorder icon on the playback window status bar. This method needs to return the source that was used for the frame around the timestamp passed via rtPos parameter. The pdwSrcIdx parameter is a 0 based source index (same one used for retrieving failover information via ILvMultiSourceSite interface as described above). The rtPos parameter indicates currently played position and is the same value that translators report back via IVideo2::GetCurrentPosition() method. It can be safely ignored for live video playback and in most cases for recorded video playback as well. It is only provided for convenience in case your player supports client side buffering – the source currently being used to read data ahead may not be the same source that was used to read data being played at the moment. Even in this case however, translator typically has the knowledge of which frame is being played back (because it is needed for correct GetCurrentPosition() implementation, so the parameter is redundant in most cases.

HRESULT EnableSource(DWORD dwSrcIdx, BOOL bEnable);

OnGuard calls this method whenever a source is marked logically online or offline (manually or automatically). These notifications can be used in order to avoid further communication with the source that was marked offline and trigger a switch to an alternative source if possible. Also, if primary recorder is marked back online, then immediate switch to the primary recorder is recommended as primary recorder is typically considered to have higher priority.
HRESULT IsSourceEnabled(DWORD dwSrcIdx, BOOL* pbEnabled);

This method is not currently utilized but that is subject to change in the future releases of OnGuard. Translator needs to keep track of the current status of each source in respect to whether or not that source is marked logically online or offline for this playback (sources can be marked offline temporarily for the playback only – when Launch Video From or Launch Recorded Video From context menu items are used in Alarm Monitoring). Basically the last value set via EnableSource() method needs to be returned here or in case EnableSource() was not called then the initial value retrieved via ILvMultiSourceSite::GetSourceInfo() method call should be returned.

3.17.2.7 IVideoRecorderConfig

This interface is optional. OnGuard Communication Server will call the IVideoRecorderConfig::Lnl_SetCamera() whenever a channel is added or modified in System Administration (described in section 3.1.1). The method contains some additional information about channel configuration in System Administration, including the failover and redundancy settings. Note that in case failover is defined for the channel being added or modified, this method will be called on both translator instances in Communication Server – one for the primary recorder and one for the secondary recorder. The meaning of some parameters is different depending on which instance receives the call as described below. Information about Lnl_SetRecorder(), another method of the interface, can be found under the section 3.12.

Note: The remaining methods of this interface, specifically Lnl_GetCameraInfo() and Lnl_GetRecorderInfo(), should simply return E_NOTIMPL.

HRESULT Lnl_SetCamera( BYTE* pCameraData, DWORD cbSize);

Information about the method can be found in sections 3.1.1 and 3.2.1. Here is an example on how to get failover and redundancy settings:

#include “lmsgtype.h”

STDMETHODIMP CVideoTrans::Lnl_SetCamera(BYTE* pCameraData, DWORD cbSize)
{
    if (cbSize < sizeof(CAMERA_DATA4))
        return E_FAIL;
    const CAMERA_DATA4* pData4 = reinterpret_cast<const CAMERA_DATA4*>(pCameraData);
    BOOL bRedundancyChecked = pData4->cd4Redundant != 0;
    int primaryRecorderID = pData4->cd4.cdPrimaryRecorderID;
    int primaryChannel = pData4->cd4.cdPrimaryChannel;
    int failoverIP = pData4->cd4.cdFailOverRecorderIP;
}

As mentioned earlier, these parameters will have a different meaning depending on which translator instance is being called – primary or failover. The bRedundancyChecked, primaryRecorderID, and primaryChannel variables in the example above will only have meaning when the translator for the failover recorder is called. primaryRecorderID can be used to separate between the instances – when its value is less than 1, then it is the translator instance for the primary recorder, otherwise it is the one for the failover recorder. For the failover recorder, bRedundancyChecked indicates the Redundancy check box setting for the channel being configured, while primaryRecorderID and primaryChannel represent the internal OnGuard panel id for the primary recorder and corresponding channel number. The failoverIP contains the address of the other recorder for both primary and recorder translator...
instances. Typically, the primary recorder’s ip, channel number and redundancy setting can be used to configure the failover recorder to start watching over the primary one. Depending on how failover or redundancy solution is implemented this information may or may not be useful.

3.18 Launch recorder web page from OnGuard

A button for launching the recorder web page can be enabled in OnGuard (on both the Video Recorder and Camera tabs – see the graphics below). The `<ConfigurationWebPage>` element should be added to the DVR Capabilities xml file with the web address of the recorder. For example:

```xml
```
4 Dynamic DVR Capabilities

This feature supports a dynamic interface that allows retrieval of information on the OAAP recorder capabilities. This feature is based on an XML file provided by the OAAP partner which includes some of the capabilities of the OAAP recorder. The XML file is stored in the OnGuard database and should be written according to the DVRCapabilities schema. This schema can be found in the OpenVideo - Recorder folder on the SDK disc. An explanation of the schema and an example of an XML capabilities file can be found below.

Notes:
- The XML file is required for integration with OnGuard only if you wish to use the features that the XML file provides (import video, video search, IntelligentVideo, etc).
- When the XML file is used in your integration, the UI_TYPE of the recorder in the PANELTYPE database should be equal to 9.

4.1 XML Schema Elements

<Recorder> - This element must be in the XML file. The element includes the recorder type and version.

<Address> - This element indicates which address fields will show up on the ‘Video Recorder-> Connection’ tab. For example, if the <IP> element is included, it means that the option to set the recorder’s IP will be available. The <PORT> element enables the option to configure port number.

<Credentials> - This element indicates if the credentials fields (user name and password) will show up on the Video Recorder > Connection tab in System Administration.

<WorldTimeZone> - This element indicates if the World Time Zone and Daylight Saving fields will show up on the Video Recorder > Connection tab in System Administration.

<Archive> - If the OAAP recorder supports integration with the archive server application, this element must be used. The element indicates if the ‘Archiving/Purging’ tab will show up on the ‘Video Recorder’ tab in System Administration. Using the element will also enable the continuous archiving UI in the ‘Camera’ tab in System Administration.

<CameraStatus> - If the OAAP recorder supports the ‘camera online/offline’ feature this element should be part of the capabilities file.

<DeleteOldVideo> - If the OAAP recorder supports the ‘automatic deletion of old video files’ feature and/or the ‘Send alarm if unable to keep video on the recorder’, then the element should be part of the capabilities file.

<Resolution> - The ‘resolution’ element should be part of the capabilities file if the OAAP recorder supports resolution setting for all the channels.

<VideoStandard> - The ‘video standard’ element should be part of the capabilities file if the OAAP recorder supports video standard (NTSC or PAL) setting for all the channels.

<CameraWizard> - The element should appear on the capabilities file if the OAAP recorder supports the ‘Camera Wizard’ feature.

<IntelligentVideo> - If the OAAP recorder supports integration with the OnGuard IntelligentVideo Server, this element must be included in the XML file. The PanelType attribute should be set with the URL prefix that will be sent to the search service.
For more information about the integration with IntelligentVideo Server can be found in section 3 OpenVideo-Recorder API Overview.

<FileAssociation> - If the OAAP recorder supports the file import feature (using the Lnl_PlayFile API) this element must be use. The element defines the file extension/s that is used in the OAAP recorders using the FileExtension attribute. The FileOfTodd attribute defines the type name of the files with the specific extension. More than one file type can be added to the XML file. The number of file types should be set on the NumOfFileTypes attribute.

<SetClock> - If the OAAP recorder supports the “set clock” feature, this element should be part of the capabilities file.

<AutoSetClockOffByDefault> - The purpose of this element is to determine the default value of the ‘Automatic clock synchronization’ checkbox (the checkbox can be found on the Recorder tab in System Administration). If the element is set in the capabilities file then the checkbox will be unchecked by default, every time an OAAP recorder is added. If the element is not included in the capabilities file, then the checkbox will be checked by default, every time an OAAP recorder is added.

<DryContacts> - This element should be part of the capabilities file if the OAAP recorder’s inputs and/or outputs need to be available for configuration from OnGuard. The element has two sub elements: <DryContactsInput/> - enables recorder’s inputs configuration; <DryContactsOutput/> - enables recorder’s outputs configuration.

<ConfigurationWebPage> - Enables the [Recorder Web Page] button on system administration UI. The value that is set for this element should be the address of the recorder web page.

<ImportCameraConfig/> - This empty element enables the Import From Recorder menu item option in the recorder context menu.

<ImportPTZPresets/> - This empty element enables PTZ presets import as part of the import feature in the recorder context menu.

<MotionDetection/> - This empty element enables the motion detection UI for channels when advanced channel configuration is used.

<LimitRecordedFrameRateByLive/> - This empty element limits the recorded frame rate by the live frame rate setting. It is available only when advanced channel configuration is used.

<SequentialRecordingFrameRates/> - This empty element enables the sequential recorded frame rate even if the camera does not support sequential live frame rate. It is available only when advanced channel configuration is used.

<CameraDirectConnect/> - This empty element enables the direct connect UI on the camera configuration page. It is available only when advanced channel configuration is used.

<CameraTimeStamps/> - This empty element enables the camera time stamps UI in the camera configuration page. It is available only when advanced channel configuration is used.

<LiveSettings/> - This empty element enables the live settings UI in the camera configuration page. It is available only when advanced channel configuration is used.
<TwoWayAudio/> - This empty element enables the two-way Audio UI in the camera configuration page. It is available only when advanced channel configuration is used.

<TurnOffCellHiding/> - This empty element disables the cell hiding feature in OnGuard. For more information, refer to section 3.5.4.

4.2 Sample XML Capabilities File

The following is an example of an XML Capabilities file:

```xml
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
<Recorder Type="OAAP" Version="8.0">
  <Address>
    <IP/>
    <PORT/>
  </Address>
  <Credentials/>
  <WorldTimeZone/>
  <Archive/>
  <CameraStatus/>
  <DeleteOldVideo/>
  <CameraWizard/>
  <IntelligentVideo PanelType="oaap"/>
  <FileAssociation NumOfFileTypes="2">
    <File FileExtension="OAAP1" FileType="The new oaap type"/>
    <File FileExtension="OAAP2" FileType="The old oaap type"/>
  </FileAssociation>
  <SetClock>
    <AutoSetClockOffByDefault/>
  </SetClock>
  <DryContacts>
    <DryContactsInput/>
    <DryContactsOutput/>
  </DryContacts>
</Recorder>
```

This sample XML file indicates the following:

- The connection for this recorder will be based on IP address and port number. There will be no field to put the DNS address.
- Credentials fields are available for this recorder type.
- Time zone field is available for this recorder. **Daylight Savings Time** check box will not be available when a recorder from this type is configured.
- The recorder supports video archiving. The Archiving UI is available in the Video Recorder and Camera tabs.
- The ‘online’ checkbox is available for cameras.
- The automatic deletion of old video files is supported by the recorder and the UI to support it is available under the ‘Capacity’ tab (under the Video Recorder tab).
- Configuration of the recorder’s resolution and video standard is not possible since the UI is not available for the user.
- The recorder supports the Camera Wizard feature. The ‘Camera Wizard’ option will be available from the recorder context menu.

- The recorder can work with the Lenel Intelligent video Server. The URL prefix that the reader will use is “oaap”.

- The recorder supports two file extensions: OAAP1 and OAAP2. When file import feature will be used from the Alarm Monitoring application, the extensions will appear along with their ‘FileOfType’ name as one of the import options.

- The recorder support clock synchronization. Every time a recorder is added, the automatic clock synchronization will be set to ‘off’.

- The recorder’s input and outputs can be used from OnGuard. Input and output panels that relates to the OAAP recorder can be configured in OnGuard.
5 OnGuard Installation and Configuration

5.1 Translator Usage

To properly implement and test OpenVideo interfaces, it is helpful to understand how the translator is used in OnGuard. There are four types of applications in OnGuard that use translator instances in distinct ways:

1. **OnGuard Communication Server**. When a recorder is configured in the System Administration application, one of the settings names the Workstation, which is the machine where Communication Server responsible for that recorder is running (or going to run). When Communication Server starts up or detects a new recorder added via System Administration, it creates a single translator instance for that recorder. This instance is used to get the status and event information from the recorder.

2. **Alarm Monitoring, VideoViewer, and System Administration**. These OnGuard clients create one or more instances of translator to view live and/or recorded video. When implementing the video playback functionality, OAAP members should keep in mind that multiple instances of the translator may be created for the same recorder, though each translator instance will always be associated with a single recorder. Also, multiple client applications and application instances may be used to view video on a single or multiple machines. In some cases, OAAP partners have a limit on number of connections to their recorders - by their SDK or licensing schema, for example. In such cases, OAAP members are responsible for enforcing these limitations, for example via implementing a singleton connection object, etc.

3. **OnGuard Linkage Server**. All OnGuard actions, such as the Select PTZ Preset action are executed in the context of this server. The Select PTZ Preset action creates an instance of the translator to use recorder based PTZ control to send the go to preset command. Multiple actions create their own instances of the translator, similar to how separate instances are created for individual video playback windows.

4. **OnGuard Archive Server**. The Archive Server is used to archive video events and continuous video off a video server. Each Archive Server creates its own instance of a video translator for each video server that it is to communicate with.

5.2 Documentation

Documentation for OnGuard installation and configuration is included with the OnGuard disc. In addition, the OnGuard SDK includes training videos for OnGuard installation and setup. The Digital Video Software User Guide provides instructions for configuring the system for video.