



Eocortex SDK and API

Table of Contents

General Information on Eocortex SDK	4
Quick start – typical tasks	5
Plugins	6
Registration of plugins in Eocortex.....	6
Action plugin.....	7
Video Analytics Plugin	10
Visualiser Plugin.....	14
Menu Item plugin	16
Event Processor plugin	18
Frame receiver plugin.....	20
Eocortex API with HTTP and RTSP interfaces	30
HTTP interface for receiving data.....	31
Receiving system configuration	32
Receiving the list of grids available in the Eocortex Client.....	43
Receiving the list of screen profiles from (views) from the server	45
Receiving information about the current screen profile in the Eocortex client	46
Receiving the current time of Eocortex server	48
Receiving information about the availability of the archive for the specified moment of time.....	49
Receiving the list of intervals with information about the beginning and end of the archive recording	50
Receiving information about the status of channels	52
HTTP interface for receiving events	56
Receiving the list of all events registered in the system.....	57
Receiving real-time events.....	58
Receiving events of the Event log	62
Receiving a list of special archive events	66
Receiving the list of recognized license plates from the archive	68
HTTP interface for executing commands by Eocortex server	70
Setting archive recording on/off for a channel.....	70
Setting date and time on Eocortex server	71
Setting a screen profile on the client	71
Setting a grid on the client	72
Setting a channel to the grid cell	72
Removing a channel from the grid cell	73
Clearing the entire grid	74
Setting channel to the guard mode	74
Sending audio to the camera	75
Generating an event from an external system	76
HTTP interface for operating PTZ features.....	78
Getting information about PTZ capabilities of the device	79

Getting device presets	81
Setting a preset	82
Getting a list of tours	83
Infrared lighting control	83
Wiper control	83
Washer control	84
"Continuous" movement	84
"Continuous" change of focus.....	84
"Continuous" zoom.....	85
Termination of "continuous" actions.....	85
Automatic focus	85
Centering.....	85
"Step-by-step" movement	86
"Step-by-step" zoom.....	86
Zooming the selected area (AreaZoom)	86
HTTP interface for receiving media data	88
Receiving a single frame	88
Receiving raw video	90
Receiving transcoded video in MJPEG format	92
Receiving an archive fragment as MP4 video file	96
RTSP interface for receiving video and sound	99
HTTP interface for managing automatic switching of views	102
Setting automatic switching profile	102
Receiving automatic switching profiles	103
Eocortex API with XML interface.....	104
Receiving People Counting data.....	105
Broadcasting video to a site.....	107
Broadcasting via HTML5	107
Broadcasting via Flash (obsolete).....	109
Broadcasting via JavaScript (obsolete)	110

General Information on Eocortex SDK

Eocortex SDK is a tool that allows you to develop software called plugins, which can extend functionality of existing **Eocortex** software system.

This tool is designed for .NET programmers who want to create plugins for **Eocortex**.

All source files of the tool and examples are coded for .NET in the C# language. **Microsoft Visual Studio** is assumed as a development environment. For understanding the document working knowledge of **Eocortex** terminology at the experienced user level is required. If necessary, you can refer to the operator and administrator instructions provided along with the **Eocortex**.



Since version 4.0, Eocortex uses [Net6](#).

All plugins developed for versions 3.6 and earlier must be reworked to be compatible with the new platform version.

Within the **Eocortex SDK** framework, each plugin software is a descendant of one of the available base classes (interfaces) from the tools, and solves a specific range of tasks. Now the main base classes (interfaces) in the tools, which can be used by external software designers, are as follows:

Plugin Name	Name	Description
ExternalAction	Action	Base class that allows adding new actions for scripts and task scheduler
VideoAnalyst	Video analytics plugin	Base class used for video analytics on the server
PluginVisualiser	Visualizer plugin	Visualizer base class used for graphic display of specific information on the Eocortex Client application channel
IClientPluginMenuItem	Menu item plugin	Base class that allows to create proper sub-item in Setup menu of Eocortex Client
EventProcessor	Event processor plugin	Event processor base class that allows to register and generate its own events, get events from Eocortex , and execute commands in the channel. Plugins of this type are used to perform integration with other systems.
ICameraServiceProvider	Frame receiver plugin	IP devices' frame receiving interface that allows to get video, audio and motion detection data and control PTZ cameras.

All the above specified base classes (interfaces) types, as well as some other supporting entities, are the subjects of related chapters of the document. All the plugins exist and operate within the **Eocortex** channel. Thus, all plugin instances are isolated from each other by default, but, if necessary, relevant data can be shared within plugin static fields. As a rule, plugins that solve the same complex task are in one .NET assembly. Such assembly is a dynamic link library (DLL) which operates within **Eocortex**. Assemblies connection and plugin registration is carried out at the stage of startup of software system separate components (see [Plugin registration in Eocortex](#)).

Quick start – typical tasks

1. IP camera integration

To connect an IP device (a camera), just implement the **Frame Receiver** plugin. All the necessary information about this type of plugin can be found in the [Frame Receiver Plugin](#) section. The plugin framework is in the folder with examples, in the **Camera.csproj** project.

2. Integration with Access Control Systems, Security and Fire Alarm, POS terminals, etc.

The integration can be performed through the **Event Processor** plugin, which is able to receive events from **Eocortex**, generate its own events in **Eocortex** in the process of interaction with other systems, execute commands in **Eocortex** (recording on / off, setting presets, I/O from the cameras etc.) in the channel, getting access to **Eocortex** archive. For information about this plugin type, see [Event processor Plugin](#) section. The plugin example can be found in the examples folder in the **EventProcessor.cproj** project. If **Eocortex** is required only to receive video and audio, there is an easier option, discussed in the section [HTTP interface for acquiring video](#); the example of acquiring video over HTTP is in the examples folder in the **HttpVideo.cproj** project.

3. Video Analytics

Any video processing algorithm can be implemented using the **Video Analytics** plugin. All the analyst results are represented as events that can be further interpreted by **Menu Item** and **Visualizer** plugins. These plugins are discussed in sections [Video Analytics Plugin](#), [Visualizer Plugin](#) and [Menu Item Plugin](#); the example of usage can be found in the examples folder in the **Analyst.csproj** project.

Plugins

The present chapter deals with the process of plugin registration. It also discusses in detail each type of plugin. The most important code fragments are shown in the text.

Registration of plugins in Eocortex

As **Eocortex** software system separate components (**Server/Client/Configurator**, hereinafter - **host**) start up, .NET assemblies are searched in the **Plugins** folder of the starting application. The **IPlugin** interface must be implemented in each found assembly, as follows:

```
public interface IPlugin
{
    /// <summary>
    /// Returns unique module identifier
    /// </summary>
    Guid Id { get; }

    /// <summary>
    /// Returns module name
    /// </summary>
    string Name { get; }

    /// <summary>
    /// Returns module manufacturer name
    /// </summary>
    string Manufacturer { get; }

    /// <summary>
    /// Module initialization.
    /// It is called by the host during module registration in the system.
    /// </summary>
    /// <param name="host">Host interface</param>
    void Initialize(IPluginHost host);
}
```

Example of the given interface implementation:

```
public class ModuleDef : IPlugin
{
    public Guid Id
    {
        get { return new Guid("17EE3457-8FC2-4C0F-B133-EF11D0C4F38C"); }
    }

    public string Name
    {
        get { return "Abandoned object detection"; }
    }

    public string Manufacturer
    {
        get { return "Eocortex"; }
    }

    public void Initialize(IPluginHost host)
    {
        host.RegisterAnalyst(typeof(AnalystExample));
        host.RegisterExternalEvent(typeof(ObjectLeftEvent));
    }
}
```

```
}  
}
```

Once the specified interface has been detected by the host, the initialization member (**Initialize**) is called. As an argument **IPluginHost** interface, providing host service methods, is transferred to the initialization member:

```
public interface IPluginHost  
{  
    /// <summary>  
    /// Gets log management interface  
    /// </summary>  
    /// <returns></returns>  
    IMcLogMgr GetLogManager();  
  
    /// <summary>  
    /// Device registration.  
    /// </summary>  
    void RegisterDevType(DevType_RegInfo regInfo);  
  
    /// <summary>  
    /// Frame receiver registration.  
    /// </summary>  
    void RegisterRTFR(RTFR_RegInfo regInfo);  
  
    /// <summary>  
    /// Registers external event  
    /// </summary>  
    void RegisterExternalEvent(Type eventType);  
  
    /// <summary>  
    /// Registers external action  
    /// </summary>  
    void RegisterExternalAction(Type actionType);  
  
    /// <summary>  
    /// Registers a menu item in the Eocortex client  
    /// </summary>  
    void RegisterMenuItem(Type menuItemType, List<Guid> requiredPluginIDs);  
  
    // ... Other registration functions not shown here  
}
```

Host service methods provide the following possibilities:

- Plugin actions logging with the **IMcLogMgr** interface, received by the **GetLogManager()** relevant method.
- Registration of plugins in the system for solving various problems.

Action plugin

An action plugin allows to extend the list of functions in scripts and in the task scheduler. It is necessary to make a class *ExternalAction* descendant to create this plugin type:

```
/// <summary>  
/// Base action class.  
/// </summary>  
[Serializable]  
public abstract class ExternalAction : IAction  
{  
    [NonSerialized]
```

```

protected IActionHost actionHost;

/// <summary>
/// Initialization. It is called by host before starting work.
/// </summary>
/// <param name="host"></param>
public virtual void Initialize(IActionHost host)
{
    actionHost = host;
}

/// <summary>
/// User action setting element. It is called by configurator.
/// Must return UserControl type (WPF).
/// </summary>
public abstract object GetGUISettingsControl();

/// <summary>
/// Displays if the current action
/// is configured properly
/// </summary>
public abstract bool IsConfigured
{
    get;
}

/// <summary>
/// The feature shows if the action is related to
/// the specific channel. In other words, whether the action affects
/// the channel in any way.
/// </summary>
public virtual bool IsChannelIndependent
{
    get
    {
        //the action is not related to channel in any way by default
        return true;
    }
}

/// <summary>
/// Starting action to be executed
/// </summary>
public abstract void Run(RawChannelEvent channelEvent);

/// <summary>
/// Command execution in the channel. It is completed by server, can be used in
/// the Run method (see above)
/// </summary>
[NonSerialized]
public ExecuteCommandDelegate ExecuteCommand;
}

```

In the descendant class the following attributes must be created:

- **ActionGUIName** – name of action, used in graphic interface in the configurator.
- **GuidAttribute** – action identification.

ActionNeedsEventArgument attribute can be determined as an option to define that the event object must be transmitted from the server to call **Run** plugin method. It also means that the action is executed only at an event, and cannot be executed in the scheduled tasks when events do not occur in these tasks.

Also, the base class methods must be defined (redefined). Let's consider in detail the initialization member in which **IActionHost** interface is passed from the host. The interface is as follows:

```
/// <summary>
/// Interface providing the host capabilities
/// when initializing the action plugin.
/// </summary>
public interface IActionHost
{
    /// <summary>
    /// Getting information about the channel,
    /// in which the current action plugin
    /// runs.
    /// </summary>
    RawChannelInfo GetChannelInfo();

    /// <summary>
    /// Saves any serialized object in the
    /// Eocortex configuration. It is used in the configurator.
    /// </summary>
    /// <param name="id">Object identifier</param>
    /// <param name="obj">Object</param>
    void SaveObject(Guid id, object obj);

    /// <summary>
    /// Gets a previously serialized object from the configuration.
    /// It is used in the configurator.
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    object GetObject(Guid id);
}
```

The interface allows to get the information on the channel to which the plugin instance is attached. The information includes the channel name and identification. The identification must be used at command execution in the channel (ref. **ExecuteCommand** delegate).

For more detailed information on available commands refer to Section [Event Processor Plugin](#). In addition, with the help of **SaveObject** and **GetObject** methods the given interface allows to save and download any serializable object using its identifier when **Eocortex** configuration is set. This technique allows to save and retrieve settings that are the same for all plugin instance configurations.

GetGUISettingsControl method must return the **UserControl** type element, which contains a graphic interface for the plugin instance configuration in the configurator. The whole graphic interface must be executed with WPF (Windows Presentation Foundation). **IsConfigured** property shows if the action is configured correctly in the graphic interface. If anything is wrong, the configuration cannot be applied unless the mistakes are corrected.

For **Action** registration in the system, it is required to call **RegisterExternalAction** host interface method (see [Plugin registration in Eocortex](#)) when downloading the assembly and plugin in the **IPlugin** initialization class method.

Video Analytics Plugin

This plugin type analyses video stream frame-by-frame, and allows to create service detectors, such as abandoned objects detector, sabotage detector and others. To create this plugin type, the **VideoAnalyst** base class descendant must be created:

```
/// <summary>
/// Video analytics class. It is used for frame and motion map processing.
/// </summary>
public abstract class VideoAnalyst : IDisposable
{
    /// <summary>
    /// Analyst initialization. It is called by host before starting processing.
    /// </summary>
    /// <param name="Id">A channel identifier</param>
    /// <param name="archiveEventsReader">Archive access interface</param>
    /// <param name="mdZones">Motion detection zones.</param>
    /// <param name="settings">Analyst settings.</param>
    public abstract void Initialize(Guid Id, IArchiveEventsReader
archiveEventsReader,
        List<MDZone> mdZones, PluginSettings settings);

    /// <summary>
    /// Frames and motion maps processing method. It is called by host.
    /// </summary>
    /// <param name="image">Frame</param>
    /// <param name="motionMap">A motion map, can be null</param>
    /// <param name="background">A motion detector background. Is equal to null, if
    /// NeedBackground == false</param>
    public abstract void Process(ImageData image, MotionMap motionMap,
        BackgroundImage background);

    /// <summary>
    /// Generates a previously registered external event in the channel.
    /// It is completed by host. It is called by analyst.
    /// </summary>
    public GenerateEventDelegate GenerateEvent;

    /// <summary>
    /// If the analyst supports work with partially decoded frames.
    /// True means that zoomed out video frames can be transmitted to the input,
    /// False means that video frames in original resolution are always transmitted
    /// to the input.
    /// </summary>
    public virtual bool SupportsPartlyDecodedFrames
    {
        get
        {
            return false;
        }
    }

    public virtual bool NeedBackground
    {
        get
        {
            return false;
        }
    }

    /// <summary>
    /// Pixel format supported

```

```

/// </summary>
public virtual VAPixelFormat PixelFormat
{
    get
    {
        return VAPixelFormat.BGR24;
    }
}

/// <summary>
/// Executes a command.
/// </summary>
/// <param name="cmdObj"></param>
public abstract object ProcessCommand(object cmdObj);

/// <summary>
/// Replaces a motion detector for a preset one in the channel.
/// It is completed by host. Can be null.
/// </summary>
public ReplaceMotionDetectorDelegate ReplaceMotionDetector;

/// <summary>
/// Resource deallocation
/// </summary>
public abstract void Dispose();

/// <summary>
/// Changes general or specific analyst settings, if necessary.
/// A calling of method shall result in opening of user setting window.
/// It is called by host (configurator).
public abstract PluginSettings SetSettings(ISettingsHost settingsHost,
    PluginSettings settings);
}

```

Descendant class must redefine all base class abstract methods and, if necessary, virtual properties. Also, a subclass must contain **PluginGUINameAttribute**, which contains the name of the analyst displayed in **Eocortex Configurator** graphical shell. In case the analyst can be set up in the configurator, implementing **SetSettings** adjustment method and specifying **PluginHasSettingsAttribute** is required.

Video data is transmitted to the analyst input as a class described below:

```

public class ImageData
{
    public DateTime Timestamp;
    public byte[] Data;
    public System.Drawing.Size Size;
    public int Stride;
    public int BitPerPixel;
}

```

As each analytic plugin is attached by the user to a channel, **PluginSettings** object of configuration in **Eocortex** configurator consists of 2 parts:

- 1) Settings related to the current security channel
- 2) General analyst settings, unrelated to the selected security channel.

```

public struct PluginSettings
{

```

```

/// <summary>
/// Specific plugin settings related to the current channel. Can be null.
/// A setting object must be serializable.
/// </summary>
public object channelSpecificSettings;

/// <summary>
/// General plugin settings. Are not related to the current channel. Can be null.
/// A setting object must be serializable.
/// </summary>
public object generalSettings;
}

```

If analyst settings are unified for all channels, it is sufficient to complete only a general setting object. Otherwise, if all the analyst settings are related to a specific channel, it is sufficient to complete only a specific channel setting object. The objects of general and specific settings are user-defined. The only requirement for them is the possibility of their serialization, as all the analyst settings are kept in **Eocortex** general configuration.

Let us consider **ProcessCommand** method which allows the analyst to execute commands from the **Menu Item** plugin (see [Menu Item plugin](#)). This method receives a command as a serialized object formed on the **Menu Item** plugin side. As a result, this method shall also return the serialized object, which will be received subsequently and processed by the **Menu Item** plugin. This mechanism allows to implement client-server interaction, as the **Videoanalyst** plugin operates on the server, and the **Menu Item** plugin always operates in the client.

While processing video frames, the videoanalyst shall transmit the results of its analysis to the server by means of event generating. To do so, it is required to register in advance on the host side (see [Plugin registration in Eocortex](#)) the outside user event containing a description of all fields required for interpreting the operating results of the analyst. The event is registered by **IPluginHost** interface using **RegisterExternalEvent** method during the module initialization stage. The user event must inherit **RawChannelEvent** base class:

```

/// <summary>
/// Parent class in event hierarchy on channel.
/// </summary>
[Serializable]
public abstract class RawChannelEvent
{
    /// <summary>
    /// Event time stamp.
    /// </summary>
    public DateTime EventTime;

    /// <summary>
    /// An event comment.
    /// </summary>
    public string Comment;

    /// <summary>
    /// If event is local.
    /// Local events are not sent outside the current host.
    /// </summary>
    public bool IsLocal = false;

    /// <summary>
    /// If the event is to be saved in the database
    /// </summary>
    public bool Save = true;
}

```

```

/// <summary>
/// The event saving mode in the database.
/// </summary>
public abstract EventArchiveSaveMode SaveMode
{
    get;
}

public RawChannelEvent()
{
    EventTime = DateTime.UtcNow;
}
}

```

Each user event should have a number of mandatory attributes:

- **GuidAttribute** – explicitly defines a unique event
- **Serializable** - allows to complete event serialization

In addition, there are following optional attributes:

- **EventNameGUI** - names the event in the host graphic interface. Unless the attribute is specified, the event is not displayed in scripts in the configurator;
- **EventNameDatabase** - a database table in which event instances are saved; the attribute must be used if the event has fields that must be saved in the database (it is important that **SaveMode** event property takes **Special** or **Both** values);
- **EventGeneratesAlarmByDefault** - a default event is an alarm, “**Generate alarm**” is automatically attached to the event when new channel is created in **Eocortex** configurator;
- **EventGenerationFrequency** - denotes event generation frequency, requires **EventGenerationFrequencyMode** (ref. below). The attribute is recommended as it affects server operation when events are saved in the database. Unless the attribute is not defined, **Middle** mode is used by default. This mode is preferred. **Low** mode is not advisable as events are recorded in the specific database, critical for functioning.

```

/// <summary>
/// Event generation frequency.
/// </summary>
public enum EventGenerationFrequencyMode
{
    /// <summary>
    /// Events are generated at frequency,
    /// close to the frequency of frame analysis
    /// </summary>
    High = 0,
    /// <summary>
    /// Events are generated at frequency compared to
    /// half of frame analysis frequency
    /// </summary>
    Middle
}

```

If the user event contains fields that must be saved in the database, the **EventFieldSaveable** attribute should be denoted for each field. The attribute needs a field ordinal number - **Order** (starting from 0) and **IsIndexable** flag, which denotes if the field index is required.

The following event field types with the following attributes **int**, **bool**, **long**, **double**, **DateTime**, **string**, **Guid**, **byte[]** are supported.

SaveMode property defines if the event is to be saved in the database. The event can be saved in a base table, that contains only **RawChannelEvent** class fields, and in a specific one with all event fields. It is possible to save all events in both tables. The base table allows to record the occurrence of the event in the system. Afterwards the user can read the events from the base table with **Eocortex** tools.

Thus, the user event can be implemented as follows:

```
[GuidAttribute("389EDCE2-54BB-4C2C-9984-51B7516A5DDF")]
[EventNameGUI("The object is left")]
[EventDatabaseName("objectleft")]
[EventGeneratesAlarmByDefault]
[EventGenerationFrequency(EventGenerationFrequencyMode.Low)]
[Serializable]
public class ObjectLeftEvent : RawChannelEvent
{
    [EventFieldSaveable(0, true)]
    [EventFieldNameGUI("Object")]
    private string objectName;

    public ObjectLeftEvent(string objectName)
    {
        this.objectName = objectName;
    }

    public override EventArchiveSaveMode SaveMode
    {
        get { return EventArchiveSaveMode.Both; }
    }
}
```

For **VideoAnalyst** plugin registration, the host interface **RegisterAnalyst** method must be called at the stage of assembly and plugin loading in **IPlugin** interface using class initialization method (see [Plugin registration in Eocortex](#)).

Visualiser Plugin

This plugin allows to display information from events that are received by **Eocortex Client** channels in graphics (e.g., frames of moving objects, identified numbers, faces, etc.). For this plugin type, the **RTVisualiser** class descendant is to be created:

```
/// <summary>
/// Visualiser class.
/// </summary>
public abstract class RTVisualiser
{
    /// <summary>
    /// Panel for primitive, text and another channel information drawing.
    /// </summary>
    protected IDrawingPanel drawingPanel;

    /// <summary>
    /// Graphical elements container. Allows to deposit separate
    /// UserControl elements in the channel.
    /// </summary>
    protected Panel controlsContrainer;

    protected IPluginToolSet pluginToolset;

    /// <summary>
    /// Visualiser initialization. It is called by host.
    /// </summary>
}
```

```

/// <param name="Id">A channel identifier</param>
/// <param name="pluginToolset"> Archive access interface used for sending
commands
/// for event subscription in the system.</param>
/// <param name="drawingPanel"> A drawing panel.</param>
/// <param name="controlsContrainer"></param>
public virtual void Initialize(Guid Id, IPluginToolSet pluginToolset,
    IDrawingPanel drawingPanel, Panel controlsContrainer)
{
    this.pluginToolset = pluginToolset;
    this.drawingPanel = drawingPanel;
    this.controlsContrainer = controlsContrainer;
}

/// <summary>
/// Visualiser event processing. Specific drawing of event results.
/// </summary>
/// <param name="channelId">A channel identifier</param>
/// <param name="chEv">Event.</param>
/// <param name="isAlarm">If event is alarm</param>
public abstract void ProcessEvent(Guid channelId, RawChannelEvent chEv,
    bool isAlarm);

/// <summary>
/// Delegate for sub-item registration in channel pop-up menu
/// in Eocortex client.
/// It is completed by host. It is called by visualiser.
/// </summary>
/// <param name="item"></param>
public RegisterChannelMenuItemHandler RegisterChannelMenuItem;

/// <summary>
/// Clearing all the visualizer drawings.
/// </summary>
public abstract void Clear();

/// <summary>
/// All resources deallocation. It is obligatory to call Release base class
/// in reloaded descendant methods.
/// </summary>
public virtual void Release()
{
    drawingPanel = null;
    controlsContrainer = null;
    pluginToolset = null;
    RegisterChannelMenuItem = null;
}
}

```

Each **Visualiser** plugin must define **ProcessEvent** method, which receives events from the channel. All events are generated by **Eocortex** and other plugins. The plugin can visualize events of any type, and they can be filtered using **if** operator and **is** key word, for example:

```

if (chEv is CounterEvent)
{
    ...
}

```

Visualiser can register its subitem in the pop-up menu (right click on the channel in the **Eocortex Client**). This allows to alter visualiser logic according to the user preference. It is provided by **RegisterChannelMenuItem** delegate.

For Visualiser plugin registration, the host interface **RegisterRTVisualiser** method must be called at the stage of assembly and plugin loading in **IPlugin** interface using class initialization method (see [Registration of plugins in Eocortex](#)).

Menu Item plugin

The plugin of this type allows to create a proper graphic interface in **Eocortex Client**, and the user can call it by clicking the corresponding **Configuration** menu item.

Typical plugin application is organization of client-server interaction with other plugins. For example, the plugin can process results of analytical plugin operation on the server. **ClientMenuItem** class descendant must be created to make a menu item plugin:

```
/// <summary>
/// Menu item class. It is used in client for adding
/// sub-items on the Configuration button menu.
/// </summary>
public abstract class ClientMenuItem : IDisposable
{
    private bool isEnabled = true;

    /// <summary>
    /// Plugin initialization.
    /// </summary>
    /// <param name="toolSet"></param>
    public abstract void Initialize(IPluginToolSet toolSet);

    /// <summary>
    /// Menu item name.
    /// </summary>
    public abstract string Name
    {
        get;
    }

    /// <summary>
    /// If the menu item is enabled.
    /// </summary>
    public bool IsEnabled
    {
        get
        {
            return isEnabled;
        }
        set
        {
            isEnabled = value;
        }
    }

    /// <summary>
    /// Processing method of click (keystroke) of the user
    /// </summary>
    public abstract void OnClicked();

    /// <summary>
    /// Resources release method.
    }
```



```

    /// </summary>
    public abstract void Dispose();
}

```

The initialization method should be defined in the descendant class, which receives interface with service functions from the host (**Eocortex Client**). The functions allow to receive channel identifiers and their names in current configuration. In addition, they provide access to **Eocortex** archive and possibility to send commands to analytical plugins and receive results of their execution. It is possible to subscribe for all events occurring in the system. The interface is as follows:

```

/// <summary>
/// A host provided interface for
/// archive access, sending commands,
/// system events subscription.
/// </summary>
public interface IPluginToolSet
{
    /// <summary>
    /// Receives archive access interface
    /// </summary>
    /// <returns></returns>
    IArchiveEventsReader GetArchiveReader();

    /// <summary>
    /// Sends a command to plugin
    /// running in the server.
    /// </summary>
    /// <param name="pluginId">Plugin identifier</param>
    /// <param name="channelsId">Channel identifiers</param>
    /// <param name="cmdObj">Command</param>
    /// <returns>Returns each channel result.
    /// The key is channel identifier.
    /// Value is a result of the command execution.</returns>
    Dictionary<Guid, object> SendChannelsCommand(Guid pluginId,
        List<Guid> channelsId, object cmdObj);

    /// <summary>
    /// Installs/deletes the channel event handler.
    /// </summary>
    /// <param name="subscrId">A subscriber identifier </param>
    /// <param name="channelId">A channel identifier </param>
    /// <param name="eventsHandler">Handler. If it is null, then
    /// previously installed handler is deleted.</param>
    void SetEventsHandler(Guid subscrId,
        Guid channelId, EventHandler eventsHandler);

    /// <summary>
    /// Receives plugin settings with a specified identifier for the channel.
    /// </summary>
    /// <param name="channelId">A channel identifier</param>
    /// <param name="pluginId">Plugin identifier</param>
    /// <returns></returns>
    PluginSettings GetPluginSettings(Guid channelId, Guid pluginId);
}

```

GetArchiveReader method provides access interface to archive and current channels information in the configuration. Detailed data about this interface are kept in **Eocortex SDK** source codes.

SendChannelsCommand method sends commands to different plugin instances of the same type according to the list of channels and receives results of command execution. **SetEventsHandler** method sets up and deletes the channel event handler.

OnClicked method must implement the user logic through graphic interface. It is called by client by clicking on the plugin-related menu item.

For **Menu Item** plugin registration, the host interface **RegisterMenuItem** should be called at the stage of assembly and plugin loading in **IPlugin** interface of class initialization method (see [Plugin registration in Eocortex](#)).

Event Processor plugin

The **Event Processor** plugin allows to receive and process signals from external systems. While processing signals, the plugin can execute commands and generate events in its channel. The event processor is created by means of inheritance from **EventProcessor** base class:

```
/// <summary>
/// Plugin class for system event processing, command and proper event generating.
/// </summary>
public abstract class EventProcessor
{
    /// <summary>
    /// Initialization. Called by host.
    /// </summary>
    /// <param name="archiveReader">Archive access interface</param>
    /// <param name="channelSpecificSettings">Plugin settings</param>
    public abstract void Initialize(IArchiveEventsReader archiveReader,
        PluginSettings settings);

    /// <summary>
    /// Generates a registered external event in the channel. Is completed
    /// by host. Called by plugin.
    /// </summary>
    public GenerateEventDelegate GenerateEvent;

    /// <summary>
    /// Sends a set command to be executed in the channel. Is completed by host.
    /// Called by plugin.
    /// </summary>
    public ExecuteCommandExDelegate ExecuteCommand;

    /// <summary>
    /// Allows to subscribe for events on the channel. Completed by
    /// plugin if necessary at the initialization stage.
    /// </summary>
    public ReceiveEventDelegate OnChannelEventReceived;

    /// <summary>
    /// Changes general or specific settings, if necessary.
    /// Method calling must result in user setting window opening.
    /// It is called by host (configurator).
    /// </summary>
    /// <param name="settings">Current plugin settings.</param>
    /// <returns>New plugin settings.</returns>
    public abstract PluginSettings SetSettings(PluginSettings settings);
}
```

A subclass must contain an obligatory attribute **PluginGUINameAttribute**, which has the analyst name displayed in **Eocortex Configurator** graphics. In case the analyst can be set up in the configurator, **SetSettings** adjustment method must be implemented and **PluginHasSettingsAttribute** identified.

Initialize method receives the interface for access to **Eocortex** archive from the host and plugin setting object. As each plugin is attached to a channel, **PluginSettings** object of configuration consists of 2 parts:

- 1) Settings related to the current security channel;
- 2) General analyst settings, unrelated to the selected security channel.

```
public struct PluginSettings
{
    /// <summary>
    /// Specific plugin settings related to the current channel. Can be null.
    /// A setting object must be serializable.
    /// </summary>
    public object channelSpecificSettings;

    /// <summary>
    /// General plugin settings. Not related to the current channel. Can be null.
    /// A setting object must be serializable.
    /// </summary>
    public object generalSettings;
}
```

If the analyst settings are unified for all channels, it will be enough to complete only general settings object. Otherwise, if all the plugin settings are related to a single channel, it will be enough to complete only a specific channel settings object. General and specific objects are user-defined. The only requirement is the object serialization, as all the plugin settings are kept in **Eocortex** general configuration.

If the plugin must execute a specific command as a result of its operation (such as, turn recording on/off, set up a preset on a camera, turn a camera, etc.), a command object should be created. Currently there are following commands:

- **RawEnableRecordingCommand** - turns on a record in the channel, record interval is defined optionally;
- **RawDisableRecordingCommand** - turns off a record in the channel;
- **RawGoToPresetPtzCommand** - sets up a preset on a camera;
- **RawGoHomePtzCommand** - sets up home camera position;
- **RawStopPtzCommand** - stops ptz (panorama/tilt/zoom) command execution on a camera;
- **RawMovePtzCommand** - one step movement;
- **RawZoomPtzCommand** - relative approximation (zoom);
- **RawStartMovePtzCommand** - continuous (preferred flowing) motion;
- **RawMoveToPtzCommand** - turns a camera so that a reference point is in the center of the frame area;
- **RawSetOutputIOCommand** - adjusts a signal level on the camera output;
- **RawSetOutputPulsesIOCommand** - generates the impulse sequence on the camera output

The **Event Processor** plugin can subscribe for events occurring at the channel. To do so the event processor has to complete **OnChannelEventReceived** delegate when initializing. In addition, the plugin can generate its own events at the channel.

For the **Event Processor** plugin registration, the host interface **RegisterEventProcessor** should be called at the stage of assembly and plugin loading in **IPlugin** interface of class initialization method (see [Plugin registration in Eocortex](#)).

Frame receiver plugin

The plugin of this type allows to receive video and sound from IP devices and motion detection data, to control PTZ cameras and IP devices' inputs/outputs (IO). To perform the subtask of receiving frames, the **IRealTimeFrameReceiver** interface must be implemented:

```
/// <summary>
/// Interface of receiving real time frames.
/// Used for creation of plugins that receive frames
/// from IP cameras.
/// </summary>
public interface IRealTimeFrameReceiver
{
    /// <summary>
    /// Event handler on receiving a new frame.
    /// It is called by the side that operates interface.
    /// The host is subscribed for events.
    /// </summary>
    event NewRawFrameEventHandler NewRawFrame;

    /// <summary>
    /// Event handler. It is called by the side that operates interface.
    /// The host is subscribed for the handler.
    /// </summary>
    event NewRawEventHandler NewEvent;

    /// <summary>
    /// Handler of a new record entering in the log. Informs the host that
    /// Connection Log field has changed (ref. below).
    /// The log is viewed in the Configurator after clicking the "Log History"
    /// It is called by the side that operates interface.
    /// The host is subscribed for the handler.
    /// </summary>
    event EventHandler NewLogRecord;

    /// <summary>
    /// A flag that denotes if it is necessary to enter a record in log.
    /// It is changed by host.
    /// </summary>
    bool IsWritingConnectionLog
    {
        get;
        set;
    }

    /// <summary>
    /// Current content of the connection log.
    /// </summary>
    string ConnectionLog
    {
        get;
    }

    /// <summary>
    /// If the stream is active
    /// </summary>
    /// <param name="streamType">Stream type</param>
    /// <returns></returns>
    bool IsStreamActive(ChannelStreamTypes streamType);

    /// <summary>
    /// Starts the specified stream to receive frames

```

```

    /// </summary>
    /// <param name="streamType"></param>
    void StartStream(ChannelStreamTypes streamType);

    /// <summary>
    /// Stops the specified stream
    /// </summary>
    /// <param name="streamType"></param>
    void StopStream(ChannelStreamTypes streamType);

    /// <summary>
    /// Sends sound to the device (duplex sound realization).
    /// </summary>
    /// <param name="soundData"></param>
    void SendSound(byte[] soundData);

    /// <summary>
    /// Releases all resources. Closes all streams.
    /// </summary>
    void Release();
}

```

While implementing this interface, it is necessary to create a mechanism for working with data streams, which can be a sequence of particular frame type, or a sequence of events. Current **Eocortex SDK** version contains the following data stream types:

```

    /// <summary>
    /// Channel stream types.
    /// </summary>
    public enum ChannelStreamTypes : ulong
    {
        /// <summary>
        /// Main video stream
        /// </summary>
        MainVideo = 1,

        /// <summary>
        /// Alternative video stream
        /// </summary>
        AlternativeVideo = 2,

        /// <summary>
        /// Camera sound stream.
        /// </summary>
        MainSound = 4,

        /// <summary>
        /// Alternative sound stream
        /// </summary>
        AlternativeSound = 8,

        /// <summary>
        /// Reverse sound stream.
        /// </summary>
        OutputSound = 16,

        /// <summary>
        /// Motion detection data stream.
        /// </summary>
        MotionDetection = 32,

        /// <summary>

```

```

        /// Camera I/O system data stream
        /// </summary>
        IO = 64,
    }

```

The data streams of different types are to be generated depending on the device capabilities and the channel settings in the configurator.

MainVideo and **AlternativeVideo** data streams consist of **RawVideoFrame** video frames, received from the camera.

MainSound and **AlternativeSound**, and **OutputSound** data streams consist of **RawSoundFrame** sound frame sequence. **MotionDetection** data stream consists of **RawChEv_MDresults** and **Raw ChEv_NoDetection** events sequence.

I/O stream consists of **RawChEv_InputSignalLevelChanged** events.

StartStream method starts the data streams from the host, in which the next stream is initialized. The method has to immediately return control to the host and implement long-term operations (input/output operations) in separate streams. Stopping the streams and control of their activities using **StopStream** and **IsStreamActive** methods is called by the host, and these actions must be completed immediately without any long-term operations. Streams return the results of their operation to the host by calling frame (**NewRawFrame**) and event (**NewEvent**) handlers.

For example, if the IP-device sends MJPEG frames, **MainVideo** (or **AlternativeVideo**), the data stream must call **NewRawFrame** and transmit **RawMJPEGFrame** video frame as one of the arguments:

```

/// <summary>
/// MJPEG frame
/// </summary>
[Serializable]
public class RawMJPEGFrame : RawVideoFrame
{
    public RawMJPEGFrame() { }

    /// <summary>
    /// The frame data
    /// </summary>
    /// <param name="data"></param>
    public RawMJPEGFrame(byte[] data)
    {
        Data = data;
    }
}

```

Likewise, for **MainSound** (or **AlternativeSound**) data stream and sound in the G.711U format, the frame **RawG711UFrame** is to be transmitted:

```

/// <summary>
/// G.711U frame
/// </summary>
[Serializable]
public class RawG711UFrame : RawSoundFrame
{

    /// <summary>
    /// The frame data
    /// </summary>
    /// <param name="data"></param>
    public RawG711UFrame(byte[] data)
    {
        this.Data = data;
        this.samplesRate = 8000;
    }
}

```

```

        this.bitsPerSample = 16;
        this.channels = 1;
        this.bitrate = 64000;
    }
}

```

After the creation of the appropriate type video frame, completing the following fields of **RawFrame** base class is required:

- **Id** – the frame identifier consisting of 2 parts: a sequence identifier (is generated at random before receiving the data stream) and a frame ordinal number.
- **Timestamp** – the frame timestamp, defined in UTC format.

In MPEG-4 and H.264 codecs:

- It is obligatory for P-frames to complete **Dependencies** field containing all frame identifiers on which the given frame depends.
- The decoder initializing information is to be defined in **SpecInitData** field in each I-frame.

Example of MJPEG frame creating:

```

RawMJPEGFrame jpegFrame = new RawMJPEGFrame(frameData);
jpegFrame.Id.SeqId = videoSeqId;
jpegFrame.Id.NumInSeq = videoNumInSeq++;
jpegFrame.TimeStamp = DateTime.UtcNow;

```

where the initial settings are:

```

// video frame sequence identifier
Guid videoSeqId = Guid.NewGuid();
// Next video frame number in the sequence
long videoNumInSeq = 0;

```

Example of H.264 I frame:

```

RawH264_I_Frame iFrame = new RawH264_I_Frame(frameData);
iFrame.Id.SeqId = videoSeqId;
iFrame.Id.NumInSeq = videoNumInSeq++;
iFrame.TimeStamp = DateTime.UtcNow;
iFrame.SpecInitData = initData;

```

where **initData** is the initializing information for a decoder.

Example of H.264 P frame:

```

RawH264_P_Frame pFrame = new RawH264_P_Frame(frameData);
pFrame.Id.SeqId = videoSeqId;
pFrame.Id.NumInSeq = videoNumInSeq++;
pFrame.TimeStamp = DateTime.UtcNow;
pFrame.Dependencies = frameDependencies;

```

where **frameDependencies** is the array of frame identifiers the frame depends on. In other words, it is a set of frames that must be decoded before the frame decoding.

Example of G.711U frame:

```

RawG711UFrame g711Frame = new RawG711UFrame(frameData);
g711Frame.Id.SeqId = audioSeqId;
g711Frame.Id.NumInSeq = audioNumInSeq++;
g711Frame.TimeStamp = DateTime.UtcNow;

```

If during the receiving of data streams a connection with the IP device becomes lost, **Frame Receiver** plugin must inform the host by generating **RawChEv_NoDataConnection** event with the obligatorily completed **StreamTypes** field which shows streams with the lost connection).

For registering frame receiver in the system, it is necessary to complete **DevType_RegInfo** registration information of the device and the **RTFR_RegInfo** plugin registration information. **DevType_RegInfo** class is described below:

```
/// <summary>
/// The device registration information.
/// is used during frame receiving plugin
/// registration.
/// </summary>
public class DevType_RegInfo
{
    /// <summary>
    /// Device identifier.
    /// </summary>
    public Guid DeviceTypeGuid;

    /// <summary>
    /// Name of manufacturer.
    /// </summary>
    public string DevTypeBrandName;

    /// <summary>
    /// Device name.
    /// </summary>
    public string DevTypeModelName;

    /// <summary>
    /// List of device capabilities.
    /// </summary>
    public DevType_Capabilities Capabilities;

    /// <summary>
    /// List of available resolutions for the device.
    /// </summary>
    public List<VideoResolutions> AvailableResolutions = new
        List<VideoResolutions>();

    /// <summary>
    /// Delegate that allows host to change camera/ video server settings.
    /// </summary>
    public SetDeviceParametersDelegate SetDeviceParameters;

    /// <summary>
    /// Receiving IRealTimeFrameReceiver interface.
    /// </summary>
    public GetRTFRDelegate GetRTFR;

    /// <summary>
    /// Receiving PTZ interface.
    /// </summary>
    public GetPtzControllerDelegate GetPtzController;

    /// <summary>
    /// Receiving I/O interface.
    /// </summary>
    public GetIOControllerDelegate GetIOController;
}
```


GetRTFR delegate, called by host, must return a specific implementation of **IRealTimeFrameReceiver** interface. The delegate receives connection parameters (**ConnectionParameters**) and substream parameters (**SubStreamParameters**), which have been defined by the user in the channel configuration as arguments.

Capabilities of IP device with which the **Frame Receiver** plugin operates are described in the **Capabilities** field:

```
/// <summary>
/// List of device capabilities
/// </summary>
public enum DevType_Capabilities : ulong
{
    /// <summary>
    /// Device supports only cameras.
    /// </summary>
    SupportsCameras = 1,
    /// <summary>
    /// Device supports cameras and video servers.
    /// </summary>
    SupportsCamerasAndServers = 2,
    /// <summary>
    /// Device supports alternative stream.
    /// </summary>
    SupportsAlternativeVideoStream = 4,
    /// <summary>
    /// Parameters (resolution, fps, compression), described by
    /// SupportedDeviceParameters/SupportedExtraParameters arrays, are
    /// independent of the format
    /// of the stream (are the same for mjpeg, mpeg4, h264).
    /// </summary>
    DeviceParametersFormatIndependent = 8,
}
```

If it is required to solve a task of controlling a tilting camera or its inputs/outputs (IO), the **IPtzController** and/or **IIOController** interfaces need to be implemented:

```
/// <summary>
/// Unified interface of tilting camera control implementation.
/// </summary>
public interface IPtzController
{
    #region ----- BASE PART -----

    /// <summary>
    /// A camera initialization.
    /// </summary>
    /// <returns></returns>
    void Initialization();

    /// <summary>
    /// Returns the camera capabilities.
    /// </summary>
    /// <returns></returns>
    PtzCapabilities GetCapabilities();

    /// <summary>
    /// Returns names of camera presets.
    /// The number of elements in resulting array corresponds to the number of
    /// presets.
    /// Each preset corresponds to a number equal to the array index.
    /// </summary>
```

```

/// <returns>Names of camera presets.</returns>
string[] GetPresetsNames();

/// <summary>
/// Defines a preset by its index.
/// </summary>
/// <param name="presetIndex"> The preset index.</param>
void SetPresetPosition(int presetIndex);

/// <summary>
/// Moves the camera to the home position.
/// </summary>
void MoveToHome();

/// <summary>
/// Stops any PTZ command executing.
/// </summary>
void Stop();

/// <summary>
/// One step movement.
/// </summary>
/// <param name="panSpeed">Horisontal speed. Range from -100 to
/// 100.</param>
/// <param name="tiltSpeed">Vertical speed. Range from -100 to
/// 100.</param>
void StepMove(int panSpeed, int tiltSpeed);

/// <summary>
/// Continuous (preferred flowing) motion.
/// </summary>
/// <param name="panSpeed">Horisontal speed. Range from -100 to
/// 100.</param>
/// <param name="tiltSpeed">Vertical speed. Range from -100 to
/// 100.</param>
void ContiniousMove(int panSpeed, int tiltSpeed);

/// <summary>
/// Relative zooming in.
/// </summary>
/// <param name="step">Step from 1 to 100.</param>
void StepZoomIn(int step);

/// <summary>
/// Relative zooming out.
/// </summary>
/// <param name="step">Step from 1 to 100.</param>
void StepZoomOut(int step);

/// <summary>
/// Continuous (preferred flowing) zooming in.
/// </summary>
/// <param name="speed">Speed. Range from 1 to 100.</param>
void ContiniousZoomIn(int speed);

/// Continuous (preferred flowing) zooming out.
/// </summary>
/// <param name="speed">Speed. Range from 1 to 100.</param>
void ContiniousZoomOut(int speed);

/// <summary>
/// Returns the camera max. zooming in.

```

```

    /// </summary>
    /// <returns>The camera max. zooming in. If the function is not
    /// supported, a negative number is returned.</returns>
    double GetMaxZoomFactor();

    /// <summary>
    /// Returns the current camera zooming in.
    /// </summary>
    /// <returns>Current camera zooming in. If the function is not supported,
    /// a negative number is returned.</returns>
    double GetCurrentZoomFactor();

    /// <summary>
    /// Sets up absolute zooming in. No operation if camera does not support
    /// the function. Ref. PtzCapabilities.
    /// </summary>
    void SetZoomFactor(double factor);

    /// <summary>
    /// Sets max. zooming in.
    /// </summary>
    void ZoomTele();

    /// <summary>
    /// Sets min. zooming in.
    /// </summary>
    void ZoomWide();

    #endregion

    #region ----- SUPPLEMENTARY PART -----

    /// <summary>
    /// Turns the camera so that reference point is
    /// in the center of the frame area.
    /// </summary>
    /// <param name="point">Display point (in pixels) that is necessary to be put
    /// in the center by turning the camera.
    /// Starting point (0,0) is the upper-left corner of frame </param>
    /// <param name="frameSize">The frame size in pixels</param>
    void MoveTo(System.Drawing.Point point, System.Drawing.Size frameSize);

    /// <summary>
    /// turns and zooms the camera so that
    /// controlled rectangle takes a full frame area.
    /// If rectangle aspect ratio does not correspond to that of the frame,
    /// then zooming is executed so that the whole rectangle
    /// fits within the frame.
    /// The rectangle center fits the frame center.
    /// </summary>
    /// <param name="rect">Rectangle is set in pixels</param>
    /// <param name="frameSize">Frame size in pixels</param>
    void ShowRect(System.Drawing.Rectangle rect, System.Drawing.Size frameSize);

    #endregion
}

    /// <summary>
    /// Unified interface of camera input/output control implementation
    /// </summary>
    public interface IIIOController
    {

```

```

    /// <summary>
    /// Initialization
    /// </summary>
    /// <returns></returns>
    void Initialization();

    /// <summary>
    /// Returns the camera capabilities.
    /// </summary>
    /// <returns></returns>
    IOCapabilities GetCapabilities();

    /// <summary>
    /// Sets the defined value in the output
    /// </summary>
    /// <param name="portID">Output No.</param>
    /// <param name="value">1 or 0</param>
    void SetOutput(int portID, int value);

    /// <summary>
    /// Supplies pulse sequence (PWM) in indicated output.
    /// It is not supported by all cameras.
    /// </summary>
    /// <param name="portID">Output No.</param>
    /// <param name="pulses">Pulse array </param>
    void SetOutput(int portID, IOPulse[] pulses)
}

```

Pan-and-tilt (IO controller) capabilities must be returned using **GetCapabilities()** method to inform the host about methods of optional capabilities implemented in the interface (if the device supports them).

DevType_RegInfo registration information of the device must be defined at the stage of loading the assembly and plugin in the class initialization method which realizes **IPlugin** interface by calling **RegisterDevType** host interface method (see [Plugin registration in Focortex](#)).

Besides **DevType_RegInfo**, it is also necessary to complete the **RTFR_RegInfo** frame receiver information:

```

    /// <summary>
    /// The frame receiver registration information
    /// </summary>
    public class RTFR_RegInfo
    {
        /// <summary>
        /// Device identifier
        /// </summary>
        public Guid DeviceTypeGuid;

        /// <summary>
        /// Data stream format
        /// </summary>
        public VideoStreamFormats StreamFormat;

        /// <summary>
        /// Connection protocol used
        /// </summary>
        public NetworkConnectionTypes ConnectionType;

        /// <summary>
        /// The device capabilities using the set format StreamFormat
        /// </summary>

```

```

        public RTFR_Capabilities Capabilities;
    }

```

This information must be specified as many times as many different stream formats the IP device supports.

Similar to **DevType_RegInfo**, **RTFR_RegInfo** information must be defined at the stage of assembly and plugin loading in **IPlugin** interface of class initialization method. Call the host interface **RegisterDevType** method to complete it. Example of completing the classes is given below:

```

public void Initialize(IPluginHost host)
{
    DevType_RegInfo KingNet_KS3002MJ_Device = new DevType_RegInfo();
    KingNet_KS3002MJ_Device.DeviceTypeGuid =
        new Guid("5C49C51D-B17B-40b0-9656-6DC71DD86D90");
    KingNet_KS3002MJ_Device.DevTypeModelName = "KS3002MJ";
    KingNet_KS3002MJ_Device.DevTypeBrandName = "KingNet";
    KingNet_KS3002MJ_Device.AvailableResolutions = GetConcreteResolutions();
    KingNet_KS3002MJ_Device.Capabilities = DevType_Capabilities.SupportsCameras;
    KingNet_KS3002MJ_Device.GetPtzController = null;
    KingNet_KS3002MJ_Device.GetRTFR = (conParam, subParams) =>
    {
        RealTimeFrameReceiver rtfr = new RealTimeFrameReceiver(conParam,
            subParams);
        return rtfr;
    };
    KingNet_KS3002MJ_Device.SetDeviceParameters = (conParams, subParams) =>
    {
        return true;
    };

    host.RegisterDevType(KingNet_KS3002MJ_Device);

    RTFR_RegInfo rtfrKingNet_KS3002MJ_Mjpeg = new RTFR_RegInfo();
    rtfrKingNet_KS3002MJ_Mjpeg.DeviceTypeGuid =
        new Guid("5C49C51D-B17B-40b0-9656-6DC71DD86D90");
    rtfrKingNet_KS3002MJ_Mjpeg.StreamFormat = VideoStreamFormats.MJPEG;
    rtfrKingNet_KS3002MJ_Mjpeg.ConnectionType = NetworkConnectionTypes.UDP;
    rtfrKingNet_KS3002MJ_Mjpeg.Capabilities.SupportedStreamTypes =
        ChannelStreamTypes.MainVideo;
    rtfrKingNet_KS3002MJ_Mjpeg.Capabilities.SupportedExtraParameters =
        new DeviceParameters[] { DeviceParameters.Null, DeviceParameters.Null };
    rtfrKingNet_KS3002MJ_Mjpeg.Capabilities.SupportedDeviceParameters =
        new DeviceParameters[] { DeviceParameters.Null, DeviceParameters.Null };

    host.RegisterRTFR(rtfrKingNet_KS3002MJ_Mjpeg);
}

```

Eocortex API with HTTP and RTSP interfaces

Eocortex API lets the user to contact the server using either HTTP or RTSP interface to receive real time and archived video streams. It is also possible to use HTTP interface to obtain system information and to send commands to the system to perform certain actions.

To access resources, you must be authorized.

Authorization methods

1. Passing authorization data through GET-parameters of the request

This method involves passing authorization data through GET-parameters of the request. These parameters can be used not only in GET-requests, but also in POST and PUT-requests when working with REST API.

Parameters

login: username (e.g. root or novikov@ent.eocortex.com for AD users)

password: user password

- **MD5 hash of the password** for the internal MC user, if the password is empty, this parameter can be omitted. (you can generate it, for example, here <https://www.md5hashgenerator.com/>).
- **Base64-encoded password** for AD user

usertype: user type

- **internal:** for internal Eocortex user, or this parameter can be omitted.
- **ActiveDirectory:** for the AD user.

2. HTTP Basic authentication

This method (https://en.wikipedia.org/wiki/Basic_access_authentication) is standard, as it is supported by most browsers and various HTTP clients, such as POSTMAN, as well as libraries (HttpClient, RestSharp) for various programming languages.

To use this authorization you should add the following authorization header to HTTP-request: **Authorization: Basic Base64(login:password)**.

For example, Authorization: Basic

bm92aWtvdkBlbnQubWFjcm9zY29wLmNvbTpQbGFpbIRleHRBZFBhc3N3b3Jk.

Parameters

login: username, which must match the user's name in Eocortex.

password: user password

- **MD5 hash of the password** for the internal Eocortex user, if the password is empty, this parameter can be omitted. (you can generate it, for example, here <https://www.md5hashgenerator.com/>).
- **The password** is a pure password for the AD user. Since the entire **login:password** combination is then **Base64** encoded, the password is not encoded separately.

For Basic authorization to work for Active Directory users, the following header must be added to the request:

UserType: ActiveDirectory

Various types of requests are described below.



User password (**password** parameter) is sent as MD5 hash in upper case.

HTTP interface for receiving data

For complex interactions with the system via API and executing some of the requests listed in this guide, information about the configuration and current state of the system, which is not displayed in the **Eocortex** GUI, can be obtained. Such information can be obtained using the CGI requests listed in this section.

Such requests in general have the following format:

```
{Protocol}://{Server}:{Port}/{Resource}?login={Login}&password={Password}&{Parameter}={Parameter value}
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	–	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol . Default ports: 8080 for http ; 18080 for https
Resource	–	URI of the server resource to which the request is addressed
Login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Parameter	–	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	–	Value of the applied additional parameter



A number of requests may have the same **command** resource. In this case the type of request is specified by the **type** parameter.



Some additional parameters are mandatory for request execution.

By default, the data is received in XML format. For some requests, however, there is a possibility to return data in JSON format. To do that it is required to specify **responsetype=json** parameter.

If there is no clear indication of the possibility to return data in JSON in a request description, it is assumed that it is to be returned in XML only.

Receiving system configuration

{...} Receiving response in JSON format is supported.

Requests to the system's API may require information about the components of the current configuration of the system. Such information can be obtained by requesting the **configex** resource.

Additional request parameters:

Parameter	Default value	Description
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/configex?login=root&password=
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/configex?login=root&password=&responsetype=json
```

Example of a response in **XML** format:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration Id="b4ca3dbb-99a3-47d3-b603-2a631f03a375"
  SenderId="ec3260bd-303d-4c35-8a39-2b80289d3c20"
  Revision="72"
  Timestamp="2023-06-07T06:30:15.3279409Z"
  XMLProtocolVersion="2"
  ServerVersion="4.1.23"
  ProductType="Ultra"
  UseTimeZones="false">
  <Servers>
    <ServerInfo Id="ec3260bd-303d-4c35-8a39-2b80289d3c20"
      Name="Server 1"
      Url="192.168.200.87:8080"
      PrimaryIp="192.168.200.87"
      PrimaryPort="8080"
      PrimarySslPort="18080"
      SecondaryIp="192.168.101.223"
      SecondaryPort="8081"
      SecondarySslPort="18081"/>
  </Servers>
  <Channels>
    <ChannelInfo Id="706c4691-3d90-41e3-8789-76eb9810648f"
      Name="Camera 1"
      Description=""
      DeviceInfo="HikVision DS-2xxxxxx, DS-N2xx"
      AttachedToServer="ec3260bd-303d-4c35-8a39-2b80289d3c20"
      IsDisabled="false"
      IsSoundOn="false"
      IsArchivingEnabled="true"
      IsSoundArchivingEnabled="true"
      AllowedRealtime="true"
    >
```



```

        AllowedArchive="true"
        IsPtzOn="false"
        IsTransmitSoundOn="false"
        ArchiveMode="AlwaysOn"
        ArchiveStreamType="Main"
        ArchiveVideoFormat="H264"
        ArchiveRotationMode="None"
        IsFaceAnalystEnabled="false"
        IsPeopleCountingOn="false"
        IsObjectCountingOn="false"
        TimeZoneOffset="5">
<Streams>
  <StreamInfo StreamType="Main"
    StreamFormat="H264"
    RotationMode="None"/>
  <StreamInfo StreamType="Alternative"
    StreamFormat="H264"
    RotationMode="By90ClockwiseDegree"/>
  <StreamInfo StreamType="SecondAlternative"
    StreamFormat="MJPEG"
    RotationMode="By90AntiClockwiseDegree"/>
  <StreamInfo StreamType="ThirdAlternative"
    StreamFormat="MJPEG"
    RotationMode="By180Degree"/>
</Streams>
<UserScenarios>
  <XmlUserScenarioInfo Id="98e9e96b-db9a-4a22-94a8-67c957d7e1fc"
    Name="Manual alarm start"
    NeedConfirmation="false"/>
</UserScenarios>
<GeoPosition Latitude="53.78760846056143"
  Longitude="-1.5578699111938477"
  Azimuth="90"/>
</ChannelInfo>
</Channels>
<RootSecurityObject Id="f50f5174-1e91-40c2-8d91-ad32119f84f3">
  <ChildSecurityObjects>
    <SecObjectInfo Id="1bee6c88-fdfd-41fd-8d9c-cd8dec8b145" Name="Floor 1">
      <ChildSecurityObjects/>
      <ChildChannels>
        <ChannelId>a3c5842b-e279-4614-8dc7-9747c5e75899</ChannelId>
      </ChildChannels>
    </SecObjectInfo>
  </ChildSecurityObjects>
  <ChildChannels/>
</RootSecurityObject>
<UserGroup>
  <GridTypesAllowed>
    <GridTypes>GridType1</GridTypes>
    <GridTypes>GridType2</GridTypes>
    <GridTypes>GridType3</GridTypes>
    <GridTypes>GridType4</GridTypes>
    <GridTypes>GridType6</GridTypes>
    <GridTypes>GridType7</GridTypes>
    <GridTypes>GridType12X11</GridTypes>
  </GridTypesAllowed>
  <Id>9a8645d1-3665-4b42-b6ef-864fa8f60c64</Id>
  <Name>Administrators</Name>
  <CanConfigure>true</CanConfigure>
  <CanConfigureWorkplace>true</CanConfigureWorkplace>
  <CanShutdown>true</CanShutdown>
  <CanChangeChannelMode>true</CanChangeChannelMode>

```

```

<CanManageRec>true</CanManageRec>
<CanAccessExpertMode>true</CanAccessExpertMode>
<CanPTZ>true</CanPTZ>
<PtzPriority>Minimal</PtzPriority>
<CanReceiveSound>true</CanReceiveSound>
<CanTransmitSound>true</CanTransmitSound>
<CanAccessNewCamera>false</CanAccessNewCamera>
<CanAccessReports>true</CanAccessReports>
<CanGetTranscodedVideoFromMobileServer>true</CanGetTranscodedVideoFromMobileServer>
<CanAccessEditingAnalystPluginsInClient>true</CanAccessEditingAnalystPluginsInClient>
<CanAccessVideoViaWeb>true</CanAccessVideoViaWeb>
<CanAccessVideoViaSmartTV>true</CanAccessVideoViaSmartTV>
<CanExportVideoToAvi>true</CanExportVideoToAvi>
<CanUseArchiveExport>true</CanUseArchiveExport>
<CanReceiveMainStream>true</CanReceiveMainStream>
<IsAllForbidden>false</IsAllForbidden>
<CanAccessUnifiedLog>true</CanAccessUnifiedLog>
<CanAccessArchiveMarks>true</CanAccessArchiveMarks>
<CanAccessSearch>true</CanAccessSearch>
<CanAccessToAllUsersInUnifiedLog>true</CanAccessToAllUsersInUnifiedLog>
<CanReceiveMobilePush>true</CanReceiveMobilePush>
<MessengerCanSendMessage>true</MessengerCanSendMessage>
<MessengerCanReceiveMessages>true</MessengerCanReceiveMessages>
<CanConfigureVideowall>true</CanConfigureVideowall>
<CanBrowsingVideowall>true</CanBrowsingVideowall>
<CanAccessPlans>true</CanAccessPlans>
<CanChangePassword>true</CanChangePassword>
<CanRunUserScenarios>true</CanRunUserScenarios>
<CanAccessGis>true</CanAccessGis>
</UserGroup>
<MobileServerInfo IsEnabled="true"
  IsProxyEnabled="true"
  IsMobilePushEnabled="true"
  Port="8089"
  UsePFrames="false"
  FpsLimit="0"
  LowResolution="120 x 90"
  MiddleResolution="240 x 180"
  HighResolution="800 x 480">
  <Resolutions>
    <ResolutionInfo Width="800"
      Height="480"
      IsEnabled="true"
      FpsLimit="15"
      UsePFrames="true"
      Type="High"/>
    <ResolutionInfo Width="240"
      Height="180"
      IsEnabled="true"
      FpsLimit="4"
      UsePFrames="false"
      Type="Middle"/>
    <ResolutionInfo Width="120"
      Height="90"
      IsEnabled="false"
      FpsLimit="4"
      UsePFrames="false"
      Type="Low"/>
  </Resolutions>
</MobileServerInfo>

```

```

<RtspServerInfo IsEnabled="true"
    TcpPort="554"
    IsMjpegEnabled="true"/>
<MobileDevicesCapabilities>
  <Archive>true</Archive>
  <Ptz>true</Ptz>
  <Hls>true</Hls>
  <AppleMobilePush>true</AppleMobilePush>
  <AndroidMobilePush>true</AndroidMobilePush>
  <Profiles>true</Profiles>
  <UserScenarios>true</UserScenarios>
  <SmartAssistant>true</SmartAssistant>
  <Gis>true</Gis>
</MobileDevicesCapabilities>
<WorldMapConfig>
  <Locations Name="All locations">
    <ChildLocations>
      <Location Name="Local office"
        Latitude="53.788007776832465"
        Longitude="-1.541484296321869"
        Zoom="19"/>
      <Location Name="Head office"
        Latitude="53.787638567873074"
        Longitude="-1.543552279472351"
        Zoom="19"/>
    </ChildLocations>
    <ChildFolders>
      <FolderLocation Name="Locations in progress">
        <ChildLocations>
          <Location Name="Business center"
            Latitude="53.788853934465465"
            Longitude="-1.5443569421768188"
            Zoom="19"/>
        </ChildLocations>
      </FolderLocation>
    </ChildFolders>
  </Locations>
</WorldMapConfig>
</Configuration>

```

Example of a response in **JSON** format:

```

{
  "Id": "b4ca3dbb-99a3-47d3-b603-2a631f03a375",
  "SenderId": "ec3260bd-303d-4c35-8a39-2b80289d3c20",
  "RevNum": 72,
  "Timestamp": "2023-06-07T06:30:15.3279409Z",
  "XmlProtocolVersion": 2,
  "ServerVersion": "4.1.23",
  "ProductType": "Ultra",
  "Servers": [
    {
      "Id": "ec3260bd-303d-4c35-8a39-2b80289d3c20",
      "Name": "Server 1",
      "Url": "192.168.200.87:8080",
      "PrimaryIp": "192.168.200.87",
      "PrimaryPort": "8080",
      "PrimarySslPort": "18080",
      "SecondaryIp": "",
      "SecondaryPort": "0",
    }
  ]
}

```

```

    "SecondarySslPort": "0",
    "ConnectionString": null
  }
],
"Channels": [
  {
    "Id": "706c4691-3d90-41e3-8789-76eb9810648f",
    "Name": "Camera 1",
    "Description": "",
    "DeviceInfo": "HikVision DS-2xxxxxx, DS-N2xx",
    "AttachedToServer": "ec3260bd-303d-4c35-8a39-2b80289d3c20",
    "IsDisabled": false,
    "IsSoundOn": false,
    "IsArchivingEnabled": true,
    "IsSoundArchivingEnabled": true,
    "AllowedRealtime": true,
    "AllowedArchive": true,
    "IsPtzOn": false,
    "IsTransmitSoundOn": false,
    "ArchiveMode": "AlwaysOn",
    "Streams": [
      {
        "StreamType": "Main",
        "StreamFormat": "H264",
        "RotationMode": "None"
      },
      {
        "StreamType": "Alternative",
        "StreamFormat": "H264",
        "RotationMode": "By90ClockwiseDegree"
      },
      {
        "StreamType": "SecondAlternative",
        "StreamFormat": "MJPEG",
        "RotationMode": "By90AntiClockwiseDegree"
      },
      {
        "StreamType": "ThirdAlternative",
        "StreamFormat": "MJPEG",
        "RotationMode": "By180Degree"
      }
    ],
    "UserScenarios": [
      {
        "Id": "98e9e96b-db9a-4a22-94a8-67c957d7e1fc",
        "Name": "Manual alarm start",
        "NeedConfirmation": false
      }
    ],
    "ArchiveStreamType": "Main",
    "ArchiveVideoFormat": "H264",
    "ArchiveRotationMode": "None",
    "IsFaceRecOn": false,
    "GeoPosition": {
      "Latitude": 53.78760846056143,
      "Longitude": -1.5578699111938477,
      "Azimuth": 90.0
    },
    "IsPeopleCountingOn": false,
    "IsObjectCountingOn": false,
    "TimeZoneOffset": 5.0
  }
]

```

```

],
"RootSecObject": {
  "ChildSecObjects": [
    {
      "ChildSecObjects": [],
      "ChildChannels": [
        "a3c5842b-e279-4614-8dc7-9747c5e75899"
      ],
      "Id": "1bee6c88-fdfd-41fd-8d9c-cd8decb8b145",
      "Name": "Floor 1"
    }
  ],
  "ChildChannels": [],
  "Id": "f50f5174-1e91-40c2-8d91-ad32119f84f3",
  "Name": null
},
"UserGroup": {
  "GridTypesAllowed": [
    "GridType1",
    "GridType2",
    "GridType3",
    "GridType4",
    "GridType6",
    "GridType7",
    "GridType12X11"
  ],
  "Id": "9a8645d1-3665-4b42-b6ef-864fa8f60c64",
  "Comment": null,
  "Name": "Administrators",
  "CanConfigure": true,
  "CanConfigureWorkplace": true,
  "CanShutdown": true,
  "CanChangeChannelMode": true,
  "CanManageRec": true,
  "CanAccessExpertMode": true,
  "CanPTZ": true,
  "PtzPriority": "Minimal",
  "CanReceiveSound": true,
  "CanTransmitSound": true,
  "CanAccessNewCamera": false,
  "CanAccessReports": true,
  "CanGetTranscodedVideoFromMobileServer": true,
  "CanAccessEditingAnalystPluginsInClient": true,
  "CanAccessVideoViaWeb": true,
  "CanAccessVideoViaSmartTV": true,
  "CanExportVideoToAvi": true,
  "CanUseArchiveExport": true,
  "CanReceiveMainStream": true,
  "IsAllForbidden": false,
  "CanAccessUnifiedLog": true,
  "CanAccessArchiveMarks": true,
  "CanAccessSearch": true,
  "CanAccessToAllUsersInUnifiedLog": true,
  "CanReceiveMobilePush": true,
  "MessengerCanSendMessage": true,
  "MessengerCanReceiveMessages": true,
  "CanConfigureVideowall": true,
  "CanBrowsingVideowall": true,
  "CanAccessPlans": true,
  "CanChangePassword": true,
  "CanRunUserScenarios": true,
  "CanAccessGis": true
}

```

```

},
"MobileServerInfo": {
  "IsEnabled": true,
  "IsProxyEnabled": true,
  "IsMobilePushEnabled": true,
  "Port": 8089,
  "UsePFFrames": false,
  "FpsLimit": 0,
  "LowResolution": "120 x 90",
  "MiddleResolution": "240 x 180",
  "HighResolution": "800 x 480",
  "Resolutions": [
    {
      "Width": 800,
      "Height": 480,
      "IsEnabled": true,
      "FpsLimit": 15,
      "UsePFFrames": true,
      "Type": "High"
    },
    {
      "Width": 240,
      "Height": 180,
      "IsEnabled": true,
      "FpsLimit": 4,
      "UsePFFrames": false,
      "Type": "Middle"
    },
    {
      "Width": 120,
      "Height": 90,
      "IsEnabled": false,
      "FpsLimit": 4,
      "UsePFFrames": false,
      "Type": "Low"
    }
  ]
},
"RtspServerInfo": {
  "IsEnabled": true,
  "TcpPort": 554,
  "IsMjpegEnabled": true
},
"MobileDevicesCapabilities": {
  "Archive": true,
  "Ptz": true,
  "Hls": true,
  "AppleMobilePush": true,
  "AndroidMobilePush": true,
  "Profiles": true,
  "UserScenarios": true,
  "SmartAssistant": true,
  "Gis": true
},
"WorldMapConfig": {
  "Locations": {
    "Name": "All locations",
    "ChildLocations": [
      {
        "Name": "Local office",
        "Latitude": 53.788007776832465,
        "Longitude": -1.541484296321869,

```

```

    "Zoom": 19.0
  },
  {
    "Name": "Head office",
    "Latitude": 53.787638567873074,
    "Longitude": -1.543552279472351,
    "Zoom": 19.0
  }
],
"ChildFolders": [
  {
    "Name": "Locations in progress",
    "ChildLocations": [
      {
        "Name": "Business center",
        "Latitude": 53.788853934465465,
        "Longitude": -1.5443569421768188,
        "Zoom": 19.0
      }
    ],
    "ChildFolders": []
  }
]
},
"UseTimeZones": false
}

```

Regardless of the selected data representation format, the response from the server will contain the same extensive information about the current system configuration, split into several sections.

The **Configuration** section, in addition to the structural inclusion of the other sections, contains the following elements on its own:

Parameter	Description
Id	Unique identifier of the current configuration
SenderId or RevNum	Unique identifier of the responded server
Revision	Number of the current configuration revision. The number increments by one after each application of configuration changes
Timestamp	Timestamp of the last configuration application
XMLProtocolVersion	Version of the XML protocol. At the moment the protocol number is 2. Its change in the future supposes the possibility of adding new elements and attributes to the xml-response
ServerVersion	Version of Eocortex installed on the responded server
ProductType	Type of license activated on the responded server
UseTimeZones	The state of the Consider time zones option. Possible values: true - option on, false - off

The **Servers** section contains a description of the servers belonging to the current configuration. Each server is described by the **ServerInfo** element, which contains the following elements:

Parameter	Description
Id	Unique server identifier

Name	Name of the server within the current configuration
Url	URL of the server
PrimaryIp	Main IP address of the server
PrimaryPort	Main HTTP port of the server
PrimarySslPort	Main HTTPS port of the server
SecondaryIp	Additional IP address of the server
SecondaryPort	Additional HTTP port of the server
SecondarySslPort	Additional HTTPS port of the server

The **Channels** section contains a description of the channel settings in the current configuration. Settings of each channel are described by the **ChannelInfo** element, which consists of the following attributes and subsections:

Parameter	Description
Id	Unique identifier of the channel. Can be used in other requests to specify the required camera with the channelid parameter
Name	Name of the channel within the current configuration
Description	Description of the camera. Can be set up using the REST API
DeviceInfo	Manufacturer and Model of the device specified for the channel
AttachedToServer	Unique identifier of the server to which the channel belongs
IsDisabled	State of the channel in the current configuration. Possible values: true — channel disabled, false — enabled
IsSoundOn	State of the option to receive sound from the camera on the specified channel. If this parameter is false , the IsSoundArchivingEnabled parameter can be ignored. Possible values: true — receiving sound is enabled, false — disabled
IsArchivingEnabled	State of the video data archiving option for the channel. Possible values: true - archiving enabled, false - disabled
IsSoundArchivingEnabled	State of the audio data archiving option for the channel. Possible values: true - archiving enabled, false - disabled
AllowedRealtime	Availability of watching video of this channel in real time for the user who sent the request. Possible values: true — viewing is allowed, false — denied
AllowedArchive	Availability of watching video of this channel from the archive for the user who sent the request. Possible values: true — viewing is allowed, false — denied
IsPtzOn	State of the PTZ control option of the camera within this channel. Possible values: true — control enabled, false — disabled
IsTransmitSoundOn	State of the option for transmitting sound to the camera speaker. Possible values: true — audio transmission enabled, false - disabled
ArchiveMode	Archive recording mode set for the channel. Possible values: AlwaysOn - recording is always active, OnlyManual - activates manually, BySchedule - activates according to the schedule, MDandManual - activates manually and automatically on demand from the motion detector
ArchiveStreamType	Video stream specified for archive recording. Possible values are the same as for the StreamType element
ArchiveVideoFormat	Format set for the video stream to be archived. Possible values are the same as for the StreamFormat element

ArchiveRotationMode	Image rotation settings for the stream to be archived. Possible values are the same as for the RotationMode element
IsFaceAnalystEnabled or IsFaceRecOn	State of the Face Recognition module. Possible values: true — module enabled, false — disabled
IsPeopleCountingOn	State of the People Counting module. Possible values: true — module enabled, false — disabled
IsObjectCountingOn	State of the Object Classification and Counting module. Possible values: true — module enabled, false — disabled
TimeZoneOffset	Camera time zone relative to UTC
Streams	Subsection containing settings of video data streams of the channel. The number of subsections depends on the settings of receiving streams for the given channel. Contains the following elements: <ul style="list-style-type: none"> • StreamType — type of stream. Possible values: Main — Main, Alternative — Additional 1, SecondAlternative — Additional 2, ThirdAlternative — Additional 3. • StreamFormat — format of the video stream. Possible values: H264, Hevc (corresponds to H.265), MJPEG, MPEG4_Part2, MxPeg. • RotationMode — settings of image rotation for the video stream. Possible values: None — no rotation, By90ClockwiseDegree — 90 degrees clockwise, By90AntiClockwiseDegree — 90 degrees counterclockwise, By180Degree — 180 degrees
UserScenarios	Subsection containing information about the User tasks created for the channel. Contains the following elements: <ul style="list-style-type: none"> • Id — unique identifier of the task. • Name — the name of the task within the current configuration. • NeedConfirmation — the state of the Confirmation of action option. Possible values: true — the task requires confirmation, false — the task starts without additional confirmation
GeoPosition	Subsection containing information about the camera coordinates on the map. Available only for systems with Enterprise and Ultra licenses. Contains the following elements: <ul style="list-style-type: none"> • Latitude — latitude. • Longitude — longitude. • Azimuth — azimuth

The **RootSecurityObject** section contains information about the structure of the security object tree and channel belonging. Consists of the following elements:

Parameter	Description
Id	Unique identifier of the security object
Name	Name of the object within the current configuration
ChildChannels	Identifiers of channels belonging to the object
ChildSecurityObjects	Subsection corresponding to an object nested in another object (Subfolder). In turn, it contains a list of elements the same as the root section of RootSecurityObject

The **UserGroup** section contains information about the settings of the group to which the user who made the configuration request belongs. Consists of the following elements and subsections:

Parameter	Description
Id	Unique identifier of the group
Comment	Comment added to the group within the current configuration
Name	Name of the group within the current configuration
GridTypesAllowed	List of grids available to this group to create views
—	A list of access rights to various system components. Possible values for the rights: true - the right is enabled, false - the right is disabled

The **MobileServerInfo** section contains information about the mobile server's video transcoding parameters. Consists of the following elements and subsections:

Parameter	Description
IsEnabled	State of stream transcoding mechanism of the mobile server. Possible values: true — transcoding of streams into the specified parameters is enabled, false — disabled
IsProxyEnabled	Option to redirect requests to HTTP(S) port instead of using own port of the mobile server. Possible values: true — HTTP(S) server port is used, false — native port of mobile server is used
IsMobilePushEnabled	Option to send Push notifications to mobile applications. Possible values: true — sending Push notifications is allowed, false — sending is prohibited
Port	Mobile server's own port. Used for interaction with mobile applications if IsProxyEnabled=false
UsePFrames	Use predicted frames (P-frames) when passing streams to mobile applications. Possible values: true — both intra-coded (I-) and predicted (P-) frames are transmitted, false — only intra-coded frames are transmitted
FpsLimit	Limit of the number of frames per second when transcoding a stream
LowResolution	Resolution of the low resolution stream
MiddleResolution	Resolution of the medium resolution stream
HighResolution	Resolution of the high resolution stream
Resolutions	Subsection containing detailed information about the settings for low , medim , and high quality streams. Consists of the following elements: <ul style="list-style-type: none"> • Width — frame width. • Height — frame height. • Type — stream type. Possible values: High, Middle, Low.

The **RtspServerInfo** section contains settings of the RTSP server used for receiving video streams from the system by third-party applications. Consists of the following elements:

Parameter	Description
IsEnabled	Availability of RTSP server for connections. Possible values: true - enabled, false - disabled
TcpPort	Port for connecting to the RTSP server
IsMjpegEnabled	State of the Allow Mjpeg broadcasting via RTSP option. Possible values: true — on, false — off

The **MobileDevicesCapabilities** section contains information about the interaction capabilities between mobile applications and the system.



Content of this section depends on the system version, not on its settings. The ability to use a particular option depends on whether it is supported in the installed **Eocortex** version. The value of the items in this section is always **true**.

Consists of the following elements:

Parameter	Description
Archive	Ability to view the archive of the system using a mobile app
Ptz	Ability to control PTZ camera functionality using a mobile app
Hls	Server-side support for HLS protocol
AppleMobilePush	Server ability to send Push notifications to iOS mobile apps
AndroidMobilePush	Server ability to send Push notifications to Android mobile apps
Profiles	Accessibility of views created in the system for mobile apps
UserScenarios	Ability to trigger the execution of user tasks configured for system cameras from mobile applications
SmartAssistant	Ability of interaction with the system with the EVA assistant
Gis	Mobile app access to maps and objects on them

The **WorldMapConfig** section contains settings for Locations created on maps. Available only for systems with **Enterprise** and **Ultra** licenses. Consists of the following elements and subsections:

Parameter	Description
Name	Name of a location or a group of locations
Latitude	Latitude value for the central point of the location
Longitude	Longitude value for the central point of the location
Zoom	Map zoom level for the location
ChildLocations	Subsection containing the list of locations belonging to the group
ChildFolders	Subsection containing a group of locations created within another group of locations (Subfolder). In turn, it contains a list of elements the same as the root WorldMapConfig section

Receiving the list of grids available in the Eocortex Client



Receiving response in JSON format is supported.

To get the list of grids that can be used by the user on whose behalf the client application is currently connected to the **Eocortex** server, a request to the **command** resource with the type **getgrids** is used.

Additional request parameters:

Parameter	Default value	Description
clientip	–	IP address of the device on which the client application is running. Necessary parameter

monitor	-	Monitor number in the current client application configuration (Starts from 0). Necessary parameter
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/command?type=getgrids&login=root&password=&clientip=192.168.1.2&monitor=0
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/command?type=getgrids&login=root&password=&clientip=192.168.1.2&monitor=0&responsetype=json
```

As a response, the server retrieves a list of available grids on the specified client.

Example of a response in **XML** format:

```
<ArrayOfString>
  <string>1</string>
  <string>4</string>
  <string>6_1</string>
  <string>7</string>
  <string>8_1</string>
  <string>9</string>
  <string>10</string>
  <string>13</string>
  <string>16</string>
  <string>25</string>
</ArrayOfString>
```

Example of a response in **JSON** format:

```
[
  "1",
  "4",
  "6_1",
  "7",
  "8_1",
  "9",
  "10",
  "13",
  "16",
  "25"
]
```

Where:

Parameter	Description
Value before "_"	Number of cells in the grid
Value after "_"	Configuration number



Configuration number exists only for grids that have the same number of cells, but differ in size and arrangement of these cells.

The obtained data can be used for changing the grid on the client.

Receiving the list of screen profiles from (views) from the server

{...} Receiving response in JSON format is supported.

For a list of screen profiles (more commonly known as views) created on the server, a request to the command resource with the type **getprofiles** is used.

Additional request parameters:

Parameter	Default value	Description
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/command?type=getprofiles&login=root&password=
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/command?type=getprofiles&login=root&password=&responsetype=json
```

As a response, the server returns a list of server profiles available for the user who sent the request.



A view is considered unavailable if it does not contain cameras that the user can view, and if the user has no rights to use that view.

Example of a response in **XML** format:

```
<ArrayOfMapInfo>
  <MapInfo>
    <Id>22ab7b5e-a8e0-48f9-80ff-a48217baa21f</Id>
    <Name>Office</Name>
  </MapInfo>
  <MapInfo>
    <Id>f2294ff9-783c-4b49-b5ab-a8fd0529da99</Id>
    <Name>Warehouse</Name>
  </MapInfo>
</ArrayOfMapInfo>
```

Example of a response in **JSON** format:

```
[
  {
    "Id": "22ab7b5e-a8e0-48f9-80ff-a48217baa21f",
    "Name": "Office"
  },
  {
```

```

    "Id": "f2294ff9-783c-4b49-b5ab-a8fd0529da99",
    "Name": "Warehouse"
  }
]

```

Where:

Parameter	Description
Id	Unique identifier of the profile. Can be used when sending a request to set the profile on the client
Name	Name of the profile within the current configuration


Receiving information about the current screen profile in the Eocortex client

{...} Receiving response in JSON format is supported.

To get information about the profile currently used on the selected **Eocortex** client monitor, use the request to the **command** resource with the type **getcurrentgrid**.

Additional request parameters:

Parameter	Default value	Description
clientip	–	IP address of the device on which the Eocortex client application is running. Necessary parameter
monitor	–	Number of the monitor in the current configuration of the client application. Starts at 0. Necessary parameter
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

 The **clientip** value must contain the actual **Eocortex** client address. Using localhost (127.0.0.1) will cause an error even when accessing a client installed on the server.

Example of a request for data in **XML** format:

```

http://127.0.0.1:8080/command?type=getcurrentgrid&login=root&password=&clientip=192.168.1.2&monitor=0

```

Example of a request for data in **JSON** format:

```

http://127.0.0.1:8080/command?type=getcurrentgrid&login=root&password=&clientip=192.168.1.2&monitor=0&responsetype=json

```

In response, the server returns information about the current profile, the type of grid used in it, and its contents.

Example of a response in **XML** format:

```

<ViewInfo>
  <GridType>GridType3</GridType>
  <ViewInfoId>20db65ef-17fb-4d75-9513-acb8535d552b</ViewInfoId>
  <GridCells>

```

```

<GridCellParameters>
  <ChannelId>a3785d7f-9259-456b-a64a-048342f22964</ChannelId>
  <CellIndex>0</CellIndex>
  <IsArchive>>false</IsArchive>
  <ArchiveStartTime xsi:nil="true"/>
  <ArchivePlaySpeed>1</ArchivePlaySpeed>
</GridCellParameters>
<GridCellParameters>
  <ChannelId>706c4691-3d90-41e3-8789-76eb9810648f</ChannelId>
  <CellIndex>1</CellIndex>
  <IsArchive>>true</IsArchive>
  <ArchiveStartTime>2023-06-09T14:09:43.7434941Z</ArchiveStartTime>
  <ArchivePlaySpeed>1</ArchivePlaySpeed>
</GridCellParameters>
<GridCellParameters>
  <ChannelId>00000000-0000-0000-0000-000000000000</ChannelId>
  <CellIndex>2</CellIndex>
  <IsArchive>>false</IsArchive>
  <ArchiveStartTime xsi:nil="true"/>
  <ArchivePlaySpeed>0</ArchivePlaySpeed>
</GridCellParameters>
</GridCells>
<ViewName>Office</ViewName>
</ViewInfo>

```

Example of a response in **JSON** format:

```

{
  "GridType": "GridType3",
  "ViewInfoId": "20db65ef-17fb-4d75-9513-acb8535d552b",
  "GridCells": [
    {
      "ChannelId": "a3785d7f-9259-456b-a64a-048342f22964",
      "CellIndex": 0,
      "IsArchive": false,
      "ArchiveStartTime": null,
      "ArchivePlaySpeed": 1.0
    },
    {
      "ChannelId": "706c4691-3d90-41e3-8789-76eb9810648f",
      "CellIndex": 1,
      "IsArchive": true,
      "ArchiveStartTime": "2023-06-09T14:09:43.7434941Z",
      "ArchivePlaySpeed": 1.0
    },
    {
      "ChannelId": "00000000-0000-0000-0000-000000000000",
      "CellIndex": 2,
      "IsArchive": false,
      "ArchiveStartTime": null,
      "ArchivePlaySpeed": 0.0
    }
  ],
  "ViewName": "Office"
}

```

Where:

Parameter	Description
GridType	Type of grid used for the selected profile
ViewInfoId	Unique identifier of the profile
GridCells	Subsection describing the contents of the cells of the selected profile . <ul style="list-style-type: none">• ChannelId — unique identifier of the channel placed in the cell.• CellIndex — cell number (Starts from zero).• IsArchive — state of the archive display mode in the cell. Possible values: true — the cell is in archive view mode, false — displaying real-time stream.• ArchiveStartTime — the current position in the archive in UTC.• ArchivePlaySpeed — video playback speed in the cell. Possible values: 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 60, 120. When displaying the real-time stream, the value is always 1
ViewName	Name of the profile within the current configuration



The obtained data can be used for setting the profile on the client.

Receiving the current time of Eocortex server



Receiving response in JSON format is supported.

To get information about the current time of the computer running the **Eocortex** server, a request to the **command** resource with the **gettime** type can be used.

Additional request parameters:

Parameter	Default value	Description
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml, json

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/command?type=gettime&login=root&password=
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/command?type=gettime&login=root&password=&responsetype=json
```

As a response, the server returns the current time of the computer in the UTC time zone.

Example of a response in **XML** format:

```
<string>14.06.2023 07:01:30</string>
```

Example of a response in **JSON** format:

```
"14.06.2023 07:01:30"
```


Receiving information about the availability of the archive for the specified moment of time

{...} Receiving response in JSON format is supported.

To get information about the availability of data in the archive for a certain point in time, a request to the resource command with the type findarchive can be used.

Additional request parameters:

Parameter	Default value	Description
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter
searchTime	–	Date and time for which the archive records are to be found. The time must be specified in the UTC time zone. Necessary parameter
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json
accept	application/json or application/xml	Selecting a response format



The **searchTime** parameter supports several date and time recording formats:

- The date can be written with a full ("YYYY" or "2023") or with an abbreviated year ("YY" or "23").
- Both a period (".") and a hyphen ("-") may be used to separate the number, month and year in the date.
- Date can be written starting with either the day ("DD.MM.YYYY" or "14.06.2023") or the year ("YYYY.MM.DD" or "2023.06.14").
- The date and time can be separated by a space (" ") or its code ("%20"), or by a plus ("+").



Although the response from the server contains a separator "T" between date and time, this separator is not supported in the request body.

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/command?type=findarchive&login=root&password=&channelid=a3785d7f-9259-456b-a64a-048342f22964&searchTime=14.06.2023+10:10:10
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/command?type=findarchive&login=root&password=&channelid=a3785d7f-9259-456b-a64a-048342f22964&searchTime=14.06.2023+10:10:10&responsetype=json
```

As a response, the server retrieves information about the availability of the archive for the specified time and the timestamp of the video frame nearest to the specified time.

Example of a response in **XML** format:

```
<CheckArchiveResult>  
  <HasArchive>true</HasArchive>  
  <NearestFrameTimestamp>2023-06-14T10:10:00.0988432Z</NearestFrameTimestamp>
```

</CheckArchiveResult>

Example of a response in **JSON** format:

```
{
  "HasArchive": true,
  "NearestFrameTimestamp": "2023-06-14T10:10:00.0988432Z"
}
```

Where:

Parameter	Description
HasArchive	Sign of the availability of the archive for the specified time. Possible values: true — the archive for the specified time exists, false — does not exist
NearestFrameTimestamp	Time stamp of the nearest video frame in UTC format. If HasArchive=false , the parameter will be set to null

Receiving the list of intervals with information about the beginning and end of the archive recording

{...} Receiving response in JSON format is supported.

To get information about the availability of record intervals in the archive for the specified period, use the request to the **archivefragments** resource.

Additional request parameters:

Parameter	Default value	Description
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter
fromtime	–	Date and time of the beginning of the period for which the information is requested. The time must be specified in the UTC time zone. Necessary parameter
totime	–	Date and time of the end of the period for which the information is requested. The time must be specified in the UTC time zone. Necessary parameter
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml, json



When specifying the parameters **fromtime** and **totime**, the following requirements must be complied:

- The date must be written with the full year ("2023" but not "23").
- - Only a period (".") can be used as a separator in the date.
- The date format should be DD.MM.YYYY ("14.06.2023").
- - The date and time should be separated by a space (" ") or its code ("%20"), or by a plus ("+").



Although the response from the server contains a different date format and the "T" separator between date and time, they are not supported in the request body.

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/archivefragments?login=root&password=&channelid=a3785d7f-9259-456b-a64a-048342f22964&fromtime=14.06.2023+00:00:00&totime=14.06.2023+10:00:00
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/archivefragments?login=root&password=&channelid=a3785d7f-9259-456b-a64a-048342f22964&fromtime=14.06.2023+00:00:00&totime=14.06.2023+10:00:00&responsetype=json
```

As a response, the server will return an array of entries containing the start and end time of the archive recording, for example, when the recording was made by the motion detector.

Example of a response in **XML** format:

```
<ArchiveFragmentsList>
  <Fragments>
    <ArchiveFragment>
      <Id>50b4d1f2-10ea-4511-940c-136217f2c84f</Id>
      <FromTime>2023-06-14T02:11:25.880943Z</FromTime>
      <ToTime>2023-06-14T09:13:22.7468942Z</ToTime>
    </ArchiveFragment>
    <ArchiveFragment>
      <Id>8c92e69f-2d17-4b39-a3bf-e43e4d93bc71</Id>
      <FromTime>2023-06-14T09:20:12.880447Z</FromTime>
      <ToTime>2023-06-14T14:09:43.7434942Z</ToTime>
    </ArchiveFragment>
  </Fragments>
</ArchiveFragmentsList>
```

Example of a response in **JSON** format:

```
{
  "Fragments": [
    {
      "Id": "50b4d1f2-10ea-4511-940c-136217f2c84f ",
      "FromTime": "2023-06-14T02:11:25.880943Z ",
      "ToTime": "2023-06-14T09:13:22.7468942Z "
    },
    {
      "Id": "8c92e69f-2d17-4b39-a3bf-e43e4d93bc71",
      "FromTime": "2023-06-14T09:20:12.880447Z ",
      "ToTime": "2023-06-14T14:09:43.7434942Z "
    }
  ]
}
```

Where:

Parameter	Description
-----------	-------------

Id	Unique identifier of the archive fragment
FromTime	Beginning time of the archive fragment
ToTime	End time of the archive fragment



The number of separate fragments of the archive for a given period depends on the incessancy of the archive recording for the specified channel. If the archive is maintained continuously, the response may contain only one fragment of the archive, even if the data was requested for several days.

Receiving information about the status of channels



Receiving response in JSON format is supported.

To get information about the state of channels and their options, a request to the **command** resource with the type **getchannelsstates** can be used.

Additional request parameters:

Parameter	Default value	Description
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/command?type=getchannelsstates&login=root&password=
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/command?type=getchannelsstates&login=root&password=&responsetype=json
```

As a response, the server will send information about the status of data streams used by the active channels.



The response to the request contains information only about the channels that are switched on. Information about existing, but switched off channels will not be sent.

Example of a response in **XML** format:

```
<ArrayOfChannelState>
  <ChannelState>
    <Id>a3785d7f-9259-456b-a64a-048342f22964</Id>
    <IsRecordingOn>>true</IsRecordingOn>
    <StreamsStates>
      <Stream>
        <Type>MainVideo</Type>
        <State>NoConnection</State>
      </Stream>
      <Stream>
        <Type>AlternativeVideo</Type>
        <State>Stopped</State>
      </Stream>
    </StreamsStates>
  </ChannelState>
</ArrayOfChannelState>
```

```
</Stream>
<Stream>
  <Type>MainSound</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>AlternativeSound</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>OutputSound</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>MotionDetection</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>IO</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>ArchiveVideo</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>ArchiveSound</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>SecondAlternativeVideo</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>ThirdAlternativeVideo</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>SecondAlternativeSound</Type>
  <State>Stopped</State>
</Stream>
<Stream>
  <Type>ThirdAlternativeSound</Type>
  <State>Stopped</State>
</Stream>
</StreamsStates>
<IsAnalogCameraConnected>>false</IsAnalogCameraConnected>
</ChannelState>
</ArrayOfChannelState>
```

Example of a response in **JSON** format:

```
[
  {
    "Id": "a3785d7f-9259-456b-a64a-048342f22964",
    "IsRecordingOn": false,
    "StreamsStates":
    [
      {
        "Type": "MainVideo",
        "State": "Active"
      },
      {
        "Type": "AlternativeVideo",
        "State": "Stopped"
      },
      {
        "Type": "MainSound",
        "State": "Active"
      },
      {
        "Type": "AlternativeSound",
        "State": "Stopped"
      },
      {
        "Type": "OutputSound",
        "State": "Stopped"
      },
      {
        "Type": "MotionDetection",
        "State": "Stopped"
      },
      {
        "Type": "IO",
        "State": "Stopped"
      },
      {
        "Type": "ArchiveVideo",
        "State": "Stopped"
      },
      {
        "Type": "ArchiveSound",
        "State": "Stopped"
      },
      {
        "Type": "SecondAlternativeVideo",
        "State": "Stopped"
      },
      {
        "Type": "ThirdAlternativeVideo",
        "State": "Stopped"
      },
      {
        "Type": "SecondAlternativeSound",
        "State": "Stopped"
      },
      {
        "Type": "ThirdAlternativeSound",
        "State": "Stopped"
      }
    ]
  }
]
```

```

    }
]

```

Where:

Parameter	Subparameter	Description
Id	–	Unique identifier of the channel
IsRecordingOn	–	State of channel archive recording. Possible values: true - archive recording is enabled, false - disabled
StreamStates		The state of data streams. Each stream has own Type and State
	Type	Type of data channel. Possible values: <ul style="list-style-type: none"> • MainVideo — video reception, stream Main; • AlternativeVideo — video reception, stream Additional 1; • MainSound — audio reception, stream Main; • AlternativeSound — audio reception, stream Additional 1; • OutputSound — audio output; • MotionDetection — built-in motion detector; • IO — digital inputs/outputs; • ArchiveVideo — archive video; • ArchiveSound — archive audio; • SecondAlternativeVideo — video reception, stream Additional 2; • ThirdAlternativeVideo — video reception, stream Additional 3; • SecondAlternativeSound — audio reception, stream Additional 2; • ThirdAlternativeSound — audio reception, stream Optional 3.
	State	State of the data channel. Possible values: <ul style="list-style-type: none"> • Stopped — the stream is stopped because it is not required by the system; • Active — the stream is in the state of receiving frames and events; • NoConnection — connection with the device was broken.
IsAnalogCameraConnected	–	Sign of the camera type used by the channel. Possible values: true — the device is connected as an analog camera, false — as a digital device

HTTP interface for receiving events

Along with obtaining information about the current system configuration, the HTTP interface also lets you to obtain information about the events detected or created by the system.

Such requests can support two structure formats: accessing a resource with and without a request type.

Accessing a resource without specifying a request type:

```
{Protocol}://{Server}:{Port}/{Resource}?login={Login}&password={Password}&{Parameter}={Parameter value}
```

Accessing a resource with a request type:

```
{Protocol}://{Server}:{Port}/{Resource}?type={Type}login={Login}&password={Password}&{Parameter}={Parameter value}
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	–	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol. Default ports: 8080 for http ; 18080 for https
Resource	–	URI of the server resource to which the request is addressed
Type	–	Type of request to the resource, if needed
Login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Parameter	–	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	–	Value of the applied additional parameter

Receiving the list of all events registered in the system



Available in Eocortex version 2.1 and later



Receiving response in JSON format is supported.

For filtering information when making requests for events, type identifiers for events can be used. Identifiers are static and unchangeable regardless of the system, the server in it or the version of the installed product.

To get a complete list of event types that exist in the system, request the **command** resource with the type **getallregisteredevents**.

Additional request parameters:

Parameter	Default value	Description
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/command?type=getallregisteredevents&login=root&password=
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/command?type=getallregisteredevents&login=root&password=&responsetype=json
```

Example of a response fragment in **XML** format:

```
<EventInfo>
  <Id>00000000-0000-0000-0000-000000000033</Id>
  <GuiName>Motion</GuiName>
  <AvailabilityModes>
    <HttpEventAvailabilityMode>RealTime</HttpEventAvailabilityMode>
  </AvailabilityModes>
</EventInfo>
```

Example of a response fragment in **JSON** format:

```
{
  "Id": "00000000-0000-0000-0000-000000000033",
  "GuiName": "Motion",
  "AvailabilityModes": [
    "RealTime"
  ]
}
```

Some event types are only available for retrieval in real time, so they cannot be retrieved when queried retrospectively. To determine which retrieval methods are available for the selected event type, check the **AvailabilityModes** parameter.

This parameter can have the following values:

- **RealTime** — events of this type are available for retrieval from the real-time stream.
- **SpecialArchive** — events of this type are available for retrieval from the archive.

Example of an event available for retrieval both in real time and from the archive:

```
<EventInfo>
  <Id>427f1cc3-2c2f-4f50-8865-56ae99c3610d</Id>
  <GuiName>Face recognized (Face Recognition module)</GuiName>
  <AvailabilityModes>
    <HttpEventAvailabilityMode>RealTime</HttpEventAvailabilityMode>
    <HttpEventAvailabilityMode>SpecialArchive</HttpEventAvailabilityMode>
  </AvailabilityModes>
</EventInfo>
```

Receiving real-time events



Available in Eocortex version 2.1 and later



Receiving response in JSON format is supported.

Events can be retrieved in real time as a continuous ("endless") HTTP connection.



When reading the response received, keep in mind that the information is transmitted using the following mechanisms:

- **Chunked transfer encoding** to transfer dynamically formed response body. Due to "infinity" of the request it is impossible to predict the exact size of the response body, therefore the response is transmitted with the "**Transfer-encoded: chunked**" header.
- **Newline delimited JSON streaming** to segregate transferred objects. Separation of objects in this mechanism is performed by placing next object to a new line instead of using symbolic delimiters.

To get an "endless" stream of system events, use a request to the **event** resource.

Additional request parameters:

Parameter	Default value	Description
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Optional parameter
filter	–	Unique identifier of the event. Can be obtained by running the request Obtaining the list of all events registered in the system . Optional parameter
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json
mode	–	Parameter to force the server to generate debug messages when applying the demo value. Optional parameter

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/event?login=root&password=&responsetype=json
```

Example of a response fragment in **JSON** format:

```
{
  "EventId" : "eb0bb455-b85f-4ac4-851f-f30a11797579",
  "Timestamp" : "19.10.2022 09:58:55",
  "BinaryTimestamp" : "5249703721781162729",
  "ZonedTimestamp" : "19.10.2022 09:58:55.377 +05:00",
  "EventDescription" : "Motion start",
  "IsAlarmEvent" : "False",
  "ChannelId" : "e0391a80-c921-4ffc-9a69-107fcf28e34e",
  "ChannelName" : "Camera 3",
  "Comment" : "",
  "EventType" : "Info",
  "InitiatorName" : "System"
}
{
  "EventId" : "e4b1f78d-35d6-4092-9fd8-72e66de82e01",
  "Timestamp" : "19.10.2022 09:59:11",
  "BinaryTimestamp" : "5249703721937844932",
  "ZonedTimestamp" : "19.10.2022 09:59:11.045 +05:00",
  "EventDescription" : "Motion stop",
  "IsAlarmEvent" : "False",
  "ChannelId" : "e0391a80-c921-4ffc-9a69-107fcf28e34e",
  "ChannelName" : "Camera 3",
  "Comment" : "",
  "EventType" : "Info",
  "InitiatorName" : "System"
}
```

The **filter** and **channelid** request parameters are used when the event of only a specific type and/or from a specific channel is to be retrieved. Both parameters can be used in the same request, but each parameter can only have one value at the same time.

Example of a valid request:

```
http://127.0.0.1:8080/event?login=root&password=&channelid=e0391a80-c921-4ffc-9a69-107fcf28e34e&filter=00000000-0000-0000-0000-000000000033&responsetype=json
```

Example of an invalid request:

```
http://127.0.0.1:8080/event?login=root&password=&channelid=e0391a80-c921-4ffc-9a69-107fcf28e34e&filter=00000000-0000-0000-0000-000000000033,e4b1f78d-35d6-4092-9fd8-72e66de82e01&responsetype=json
```

If the parameter **mode=demo** is added to the request, the system will start generating virtual events with the "Recognized license plate" event type. Such events are not saved to the system event log, not associated with any camera and do not contain any actual information. This parameter may be helpful for studying, testing and debugging the mechanisms of receiving and reading events.

Example of **virtual** events request in **JSON** format:

```
http://127.0.0.1:8080/event?login=root&password=&mode=demo&responsetype=json
```

Example of a **virtual** event in **JSON** format:

```
{
  "EventId" : "c9d6d086-c965-4cf8-ae6f-85b3894e3a4a",
  "Timestamp" : "19.10.2022 10:24:02",
  "BinaryTimestamp" : "5249703736847974593",
```

```

"ZonedTimestamp" : "19.10.2022 10:24:02.058 +05:00",
"EventDescription" : "Recognized license plate",
"IsAlarmEvent" : "False",
"ChannelId" : "00000000-0000-0000-0000-000000000000",
"ChannelName" : "",
"Comment" : "",
"EventType" : "Info",
"InitiatorName" : "System",
"IsIdentified" : "False",
"plateText" : "",
"Speed" : "0",
"Reliability" : "0",
"Left" : "0",
"Top" : "0",
"lastName" : "",
"firstName" : "",
"patronymic" : "",
"carbrand" : "",
"carcolor" : "",
"additionalInfo" : "",
"groups" : "",
"direction" : ["Unknown"],
"ExternalId" : "",
"ExternalOwnerId" : "",
"Width" : "0",
"Height" : "0",
"Numberplate" : ""
}

```

In case of complete or temporary absence of events fitting the request sent, the system will return **KeepAlive** messages in response to keep the established connection alive. A message of this type will be displayed in the stream even if a filter by event type was set in the request.

Example of **KeepAlive** message in **JSON** format:

```

{
  "EventId" : "e9e7a69c-7ee2-3fee-a530-9f8a88124fcc",
  "Timestamp" : "19.10.2022 09:59:19",
  "BinaryTimestamp" : "5249703722021050198",
  "ZonedTimestamp" : "19.10.2022 09:59:19.366 +05:00",
  "EventDescription" : "",
  "IsAlarmEvent" : "False",
  "ChannelId" : "00000000-0000-0000-0000-000000000000",
  "ChannelName" : "",
  "Comment" : "KeepAlive",
  "EventType" : "Info",
  "InitiatorName" : "System"
}

```

The results of some analytics modules contain the coordinates of an object detected by the system on the frame (for example, coordinates of a face for Face Recognition). When requesting events of this type, the response body includes relative coordinates of the object boundaries in the form of the position on the frame of the upper left point of the boundary (**Top, Left**), as well as its width and height (**Width, Height**).

The coordinates are defined from the upper left corner of the camera frame.

Example of a response with object coordinates:

```
{
  "InitiatorName" : "ExternalEvent",
  "EventId" : "427f1cc3-2c2f-4f50-8865-56ae99c3610d",
  "EventType" : "Info",
  "IsAlarmEvent" : "False",
  "ChannelId" : "cb07636a-ec9f-4555-a1eb-7b3485e1285e",
  "ChannelName" : "Camera 1",
  "Comment" : "",
  "Timestamp" : "21.04.2021 04:33:19.555",
  "BinaryTimestamp" : "5249231782422939709",
  "ZonedTimestamp" : "21.04.2021 04:33:19.555 +05:00",
  "EventDescription" : "Face recognized (Face Recognition module)",
  "FaceId" : "5821fc2b-c9d2-4d72-aa85-6ffe114b7fec",
  "IsIdentified" : "False",
  "lastName" : "",
  "firstName" : "",
  "patronymic" : "",
  "groups" : "",
  "additionalInfo" : "",
  "Left" : "0,0277343735098839",
  "Top" : "0,0372685134410858",
  "Width" : "0,25234375",
  "Height" : "0,448611122369766",
  "Similarity" : "0",
  "Age" : "27",
  "Gender" : "2",
  "ExternalId" : "",
  "TemperatureDegreesCelsius" : "0",
  "ImageBytes" : "",
  "Emotion" : ["Neutral"],
  "EmotionConfidence" : "0,729602694511414",
  "IsFaceCovered" : "False",
  "IsRotated" : "False",
  "TrajectoryId" : "9351b23f-813f-42a4-b1ba-011dbbcce99b"
}
```

For sorting by time of event occurrence, each object has a timestamp provided in the response body in three formats:

- **Timestamp** - server date and time in UTC format without specifying the server's time zone;
- **ZonedTimestamp** - date and time of the server in UTC format with specifying the server's time zone;
- **BinaryTimestamp** - date and time of the server in binary representation (DateTime.ToBinary method).



Binary representation of the timestamp has the greatest accuracy and ease of integration, but complicates reading of the timestamp for a person. To convert the binary representation of date and time to UTC format, use the DateTime.FromBinary method.

Obtaining data from the Recognized license plates event

The following fields are added to the event:



Implemented in Eocortex version 4.3

- Recognized brand;
- Reliability of brand recognition;
- Recognized car color;
- Reliability of color recognition;
- Recognized type.

Example of event request in **JSON** format:

```
http://127.0.0.1:8080/event?login=root&password=&responsetype=json
```

Example of a response in **JSON** format:

```
{
  "InitiatorName" : "ExternalEvent",
  "EventId" : "c9d6d086-c965-4cf8-aef6-85b3894e3a4a",
  "EventType" : "Info",
  "IsAlarmEvent" : "False",
  "ChannelId" : "7d53f293-70d9-45df-9243-afe5fac19c65",
  "ChannelName" : "Camera1",
  "Comment" : "5K15J",
  "Timestamp" : "26.03.2024 11:23:22",
  "BinaryTimestamp" : "5250156508449395668",
  "ZonedTimestamp" : "26.03.2024 11:23:22.200 +05:00",
  "EventDescription" : "Recognized license plates",
  "Numberplate" : "5K15J",
  "IsIdentified" : "False",
  "plateText" : "5K15J",
  "Speed" : "0",
  "Reliability" : "0,9999972581863403",
  "Left" : "0,740117073059082",
  "Top" : "0,8195064067840576",
  "lastName" : "",
  "firstName" : "",
  "patronymic" : "",
  "carbrand" : "",
  "carcolor" : "",
  "additionalInfo" : "",
  "groups" : "",
  "direction" : "Enter",
  "ExternalId" : "",
  "ExternalOwnerId" : "",
  "plateColorClass" : "Black",
  "plateCountryCode" : "am",
  "RecognizedCarBrand" : "Toyota",
  "RecognizedCarBrandConfidence" : "99,74390411376953",
  "RecognizedCarColors" : "Red",
  "RecognizedCarColorConfidence" : "99,77999877929688",
  "RecognizedCarType" : "Passenger car",
  "Width" : "0,04741443693637848",
  "Height" : "0,03725172206759453"
}
```

Receiving events of the Event log




Available in Eocortex version 4.0 and later



Receiving response in JSON format is supported.

Option of receiving events from the archive allows to create an own event log using the received data.

 The response to the request will contain all types of events available in the Events Log, and also the details of the Face Recognition module event - Face Recognized.

 A request consists of a URL and a request body sent in JSON format. For successful execution it is necessary that the sender application knows how to handle such requests.

To get a list of events from the archive, use a **GET** request to the **archive_events** resource.

Additional URL parameters:

Parameter	Default value	Description
shortevent	false	Parameter that forces the server to send a shortened version of the event message
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json


A basic request lets you access a URL without filling in the body of the request.

Example of URL:

http://127.0.0.1:8080/archive_events?login=root&password=

As a response to this request, a complete list of system events will be sent without filtering.

To clarify the request, add additional parameters to the request by specifying them in the request body in **JSON** format.

 If a parameter is not specified in the request body, the default value will be applied when executing the request.

Additional request body parameters:

Parameter	Type	Default value	Description
StartTimeUtc	string	Minimum possible time	Search start time in the format of dd.MM.yyyy HH:mm:ss.fff (UTC)
EndTimeUtc	string	Maximum possible time	Search end time in the format of dd.MM.yyyy HH:mm:ss.fff (UTC)
EventCategories	number	All categories	Categories of events. Available options: 0 – information 1 – alarm 2 – error
EventInitiatorTypes	number	All types of event initiators	Types of event initiators. Available options: 0 – System 1 – User 2 – Scenario 3 – User task

			4 — Scheduled task 8 — External module
EventInitiators	string	All users	List of system user identifiers (Guid)
EventIds	string	All events	List of event identifiers
ChannelIds	string	All cameras and empty Guid	List of camera identifiers To retrieve events that are not associated with a camera, send an empty Guid (00000000-0000-0000-0000-000000000000)
IsSearchFromBegin	boolean	false	Search for events in the archive from the beginning of the time interval
SearchLimitCount	number	5000	Maximum number of entries to retrieve



Sending a request with incorrect values of the parameters below will result in a response with error code **400 (Bad Request)**:

- StartTimeUtc
- EndTimeUtc
- IsSearchFromBegin
- SearchLimitCount

Example of request body with additional parameters in **JSON** format:

```
{
  "startTimeUtc": "17.04.2023 09:54:31.775",
  "endTimeUtc": "17.04.2023 10:54:31.775",
  "cameraIds": ["00000000-0000-0000-0000-000000000000"],
  "eventCategories": [0,1,2],
  "eventInitiatorTypes": [0,2,8,4,1,3],
  "eventInitiators": ["2aa0118f-8849-499c-b0df-6b071d95ee66"],
  "isSearchFromBegin": false,
  "searchLimitCount": 200
}
```

As a response, the server will send information about the event in one of the following formats, depending on the additional parameters of the request and the properties of the event itself.

Example of an event without additional information:

```
{
  "ChannelId": "00000000-0000-0000-0000-000000000000",
  "ChannelName": "",
  "Event": null,
  "EventCategory": 0,
  "EventComment": "Recording speed: 1,59 MB/s. Subsystem to work with archive.",
  "EventDescription": null,
  "EventId": "00000000-0000-0000-0000-000000000010",
  "EventInitiatorType": 0,
  "Timestamp": "2022-10-03T08:45:20.9497012Z"
}
```

Example of an event with additional information:

```
{
  "ChannelId": "effbcd69-9f89-4301-87bf-2663bff0a44d",
```



```

"ChannelName": "Camera 9",
"Event": {
  "AdditionalInfo": "Helen from the management team",
  "Age": 24,
  "Emotion": 2,
  "EventName": "FaceDetectedNotifyEvent",
  "EventTime": "2022-10-03T08:23:40.8019552Z",
  "FaceId": "fe705eb2-4f76-47eb-92eb-178c0ccf7077",
  "FaceImageBase64": "base64 jpeg image",
  "FirstName": "Helen",
  "Gender": 1,
  "Groups": ["White list", "Management"],
  "IsIdentified": true,
  "LastName": "Smith",
  "Patronymic": "",
  "Height": 0.152734375,
  "Left": 0.56064453125,
  "Top": 0.3083984375,
  "Width": 0.11455078125,
  "Similarity": 0.9900000095367432,
},
"EventCategory": 1,
"EventComment": "Helen Smith",
"EventDescription": "Face recognized (Face recognition module)",
"EventId": "427f1cc3-2c2f-4f50-8865-56ae99c3610d",
"EventInitiatorType": 0,
"Timestamp": "2022-10-03T08:23:40.8019552Z"
}

```

If the parameter **shorthevent=true** is specified in the request URL, the server will return the shortened event text.

Example of a shortened event text request:

```
http://127.0.0.1:8080/archive_events?login=root&password=&shorthevent=true
```

Example of a response with shortened event text:

```

{
  "EventCategory": 1,
  "EventId": "427f1cc3-2c2f-4f50-8865-56ae99c3610d",
  "Timestamp": "2022-10-03T08:23:40.8019552Z"
}

```

The following parameters can be found in the event text sent in the body of the response sent back by the server:

Parameter	Type	Description
ChannelId	Guid	Camera identifier
ChannelName	string	Camera name
EventId	Guid	Event type identifier
EventCategory	number	Event category: 0 — information 1 — alarm 2 — error
EventComment	string	Commentary from the event
EventDescription	string	Event description
EventInitiatorType	number	Event initiator type:

		0 – System 1 – User 2 – Scenario 3 – User task 4 – Scheduled task 8 – External module
Timestamp	string	Time of event (UTC)
Event	For Unified events - null For events of analytic modules - information about analysis results (if exists)	Attached entry with information about the event

To obtain a list of event types registered in the system, which can be sent in response to a request to the **archive_events** resource, a request to the **archive_event_types** resource is used.



Querying this resource does not involve sending an additional request body in JSON format.

Additional URL parameters:

Parameter	Default value	Description
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

Request sample for retrieving data in JSON format:

```
http://127.0.0.1:8080/archive_event_types?login=root&password=&responsetype=json
```

As a response to this request, the server will send the list of events in JSON format as follows:

```
[
  {
    "Id": "7da43299-9c0d-4043-9183-e947b5f6bdca",
    "Name": "Loud sound"
  }
]
```

Receiving a list of special archive events





Available in Eocortex version 2.1 and later



Receiving response in JSON format is supported.

Another way to get events from the archive is a batch unloading of data in the form of a list of events recorded by the system over a specified period of time. For this purpose, a query to the **specialarchiveevents** resource can be used.

-  When reading the received response, take into account that the information is transferred using **Newline delimited JSON streaming** mechanism. Separation of objects in this mechanism is performed by placing the next object on a new line instead of using symbolic delimiters.
-  Retrieval of events from the archive is available in the form of a list of events recorded by the system during a specified time. The maximum number of events per request is **1000**. To retrieve further events, it is necessary to repeat the search from the time of the last received event.

Additional request parameters:

Parameter	Default value	Description
starttime	–	Date and time of the beginning of the event search interval. Must be specified according to the UTC time zone in the format DD-MM-YYYY+hh:mm:ss. Necessary parameter
endtime	–	Date and time of the end of the event search interval. Must be specified according to the UTC time zone in the format DD-MM-YYYY+hh:mm:ss . Necessary parameter
eventid	–	Unique identifier of the event type. Can be obtained by running the query Obtaining the list of all events registered in the system . Necessary parameter
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Optional parameter
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml, json

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/specialarchiveevents?login=root&password=&starttime=19.10.2022+10:50:00&endtime=19.10.2022+11:00:00&eventid=b0536c2f-2f09-4969-bf1a-9fb847b87d21&responsetype=json
```

Example of a response in **JSON** format:

```
{
  "EventId" : "b0536c2f-2f09-4969-bf1a-9fb847b87d21",
  "Timestamp" : "19.10.2022 10:59:56",
  "BinaryTimestamp" : "5249703758396001539",
  "ZonedTimestamp" : "19.10.2022 10:59:56.861 +05:00",
  "EventDescription" : "Tracking event",
  "IsAlarmEvent" : "True",
  "ChannelId" : "e0391a80-c921-4ffc-9a69-107fcf28e34e",
  "ChannelName" : "Camera 3",
  "Comment" : "Movement in the zone",
  "EventType" : "Alarm",
  "InitiatorName" : "ExternalEvent",
  "alertType" : ["MovingInZone"],
```

```

"AlertTime" : "638017739968613635",
"TrajectoryId" : "57f88149-9976-4953-9b4c-3921c689b82d",
"Left" : "0,0032153846710991974",
"Top" : "0,5185692308091415",
"Width" : "0,05787692314846298",
"Height" : "0,13713846175798144"
}

```

Receiving the list of recognized license plates from the archive

{...} Receiving response in JSON format is supported.

The License Plate Recognition module has an additional type of query that makes it possible to get the list of license plates recognized during a specified time period in a shorter format than with the [event](#) or [specialarchiveevents](#) resources.

To get the list of recognized license plates in shortened form, the query to the **autovprs_export** resource can be used.

Additional request parameters:

Parameter	Default value	Description
starttime	–	The start time of the event search interval. Specified by server time zone in the format YYYY-MM-DD-hh-mm-ss-fff. Necessary parameter
finishtime	–	The end time of the event search interval. Specified by server time zone in the format YYYY-MM-DD-hh-mm-ss-fff. Necessary parameter
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter



The answer from the server for this request will always be in JSON format.

Example of a request for data in **JSON** format:

```

http://127.0.0.1:8080/autovprs_export?login=root&password=&channelid=f2e18694-3f58-4128-a48e-37dd184b109b&startTime=2022-10-20-16-40-00-000&finishTime=2022-10-20-17-00-00-000

```

Example of a response in **JSON** format:

```

[
  {
    "TimeUtc" : "20.10.2022 11:48:11.707",
    "Numberplate" : "120P90B",
    "LastName" : "",
    "FirstName" : "",
    "PatronymicName" : "",
    "Group" : "",
    "Direction" : "Unknown"
  },
  {
    "TimeUtc" : "20.10.2022 11:48:35.097",

```

```
"Numberplate" : "CXH1220",  
"LastName" : "",  
"FirstName" : "",  
"PatronymicName" : "",  
"Group" : "",  
"Direction" : "Unknown"  
}  
]
```



When sending a request and reading a response it is necessary to take into account the peculiarities of the time display.

The request body shall contain the time interval in the server time zone in the format YYYY-MM-DD-hh-mm-ss-ms.

In the response body, the license plate recognition time is displayed in the UTC(+0) time zone in the format DD-MM-YYYYY hh-mm-ss.

Thus, if the request is sent to the server with the UTC+3 time zone to obtain the list of recognized license plate numbers within the interval of 15:00:00 - 15:10:00, the time in the request body shall be displayed as per the required interval, while in the response body the corresponding car license numbers will be displayed within the 12:00:00 - 12:10:00 range.

HTTP interface for executing commands by Eocortex server

The **Eocortex** HTTP-interface lets you not only get information about the system, but also manage some of its components by executing the CGI requests described below.

Such requests, with one exception, have the following format:

```
{Protocol}://{Server}:{Port}/command?type={Type}&login={Login}&password={Password}&{Parameter}={Parameter value}
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	–	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol. Default ports: 8080 for http ; 18080 for https
Type	–	Type of request to the resource, if needed
Login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Parameter	–	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	–	Value of the applied additional parameter

For most of the requests, the server sends the same type of response about the status of the request. If the request was successfully executed, the response body will contain only the **OK** or **Success** status. Otherwise, the response body will contain information about an execution error.

Setting archive recording on/off for a channel

To manually switch the archive recording state for the channel, use the **recording** type of request.

Additional request parameters:

Parameter	Default value	Description
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter
mode	–	Archive recording mode. Possible values: start — start recording, stop — stop recording. Necessary parameter

interval	-	Recording time in minutes. Necessary parameter when mode=start
----------	---	---

Example of a request to turn on a recording lasting 15 minutes:

```
http://127.0.0.1:8080/command?type=recording&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&login=root&password=&mode=start&interval=15
```

Example of a request to turn off a recording:

```
http://127.0.0.1:8080/command?type=recording&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&login=root&password=&mode=stop
```



This request has no effect on channels for which archive recording is disabled or is already running continuously.

Setting date and time on Eocortex server

To set the time forcibly on the device where the **Eocortex** server is installed, use the **settime** type of request.

Additional request parameters:

Parameter	Default value	Description
time	-	Date and time in the UTC time zone. Necessary parameter



The **time** parameter supports several date and time recording formats:

- The date can be written either with a full year ("YYYY" or "2023") or with an abbreviated year ("YY" or "23").
- Both a period (".") and a hyphen ("-") may be used to distinguish the day, month, and year in the date.
- Date can be written starting with either the day ("DD.MM.YYYY" or "14.06.2023") or the year ("YYYY.MM.DD" or "2023.06.14").
- The date and time can be separated by a space (" ") or its code ("%20"), or by a plus ("+").

Example of a request:

```
http://127.0.0.1:8080/command?type=settime&login=root&password=&time=14.06.2023+08:11:00
```

Setting a screen profile on the client

To send a command to the **Eocortex** Client connected to the server to change the screen profile, use the **setprofile** type of request.



Only server-based screen profiles can be set with this request. Profiles (views) created directly on the client are not supported.

Additional request parameters:

Parameter	Default value	Description
clientip	-	IP address of the device running the Eocortex client application. Necessary parameter

monitor	–	Number of the monitor in the current configuration of the client application. Starts from 0. Necessary parameter
profileid	–	Identifier of the screen profile to be set. The list of available profiles can be got with the request for Receiving the list of screen profiles



The **clientip** value must contain the actual address of the **Eocortex** client. Using localhost (127.0.0.1) will cause an error even when accessing a client installed directly on the server.

Example of a request:

```
http://127.0.0.1:8080/command?type=setprofile&login=root&password=&clientip=192.168.1.2&monitor=0&profileid=13851f3d-c7d3-4ec6-b0ff-2d66873bf118
```

Setting a grid on the client

To change the camera grid within the current screen profile on the selected **Eocortex** client, use the **setgrid** type of request.



The request lets changing grids only for those screen profiles that were created directly on the client device. Changing server profiles is not supported.



Any changes to the grid will be saved automatically within the current screen profile. A new profile will not be created.

Additional request parameters:

Parameter	Default value	Description
clientip	–	IP address of the device running the Eocortex client application. Necessary parameter
monitor	–	Number of the monitor in the current configuration of the client application. Starts from 0. Necessary parameter
cells	–	The name of the grid to be set. Names of the available grids can be obtained using the Receiving the list of grids request



The **clientip** value must contain the actual address of the **Eocortex** client. Using localhost (127.0.0.1) will cause an error even when accessing a client installed directly on the server.

Example of a request:

```
http://127.0.0.1:8080/command?type=setgrid&login=root&password=&clientip=192.168.1.2&monitor=0&cells=25
```

Setting a channel to the grid cell

To place a new channel into a grid cell of the active screen profile, use the **setcell** type of request.



The request lets changing grids only for those screen profiles that were created directly on the client device. Changing server profiles is not supported.



Any changes to the grid will be saved automatically within the current screen profile. A new profile will not be created.

Additional request parameters:

Parameter	Default value	Description
clientip	–	IP address of the device running the Eocortex client application. Necessary parameter
monitor	–	Number of the monitor in the current configuration of the client application. Starts from 0. Necessary parameter
cell	–	Cell number. Starts from 0. Necessary parameter
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter
mode	realtime	Channel playback mode in the cell. Possible values: realtime - playback of real time stream, archive - playback of archive
starttime	Moment of receiving the request	Initial point of the archive playback. Applies only when mode=archive
speed	1	Archive playback speed. Applies only when mode=archive . Possible values: 0.1; 0.2; 0.5; 1; 2; 5; 10; 20; 60; 120



The **clientip** value must contain the actual address of the **Eocortex** client. Using localhost (127.0.0.1) will cause an error even when accessing a client installed directly on the server.



The **time** parameter supports several date and time recording formats:

- The date can be written either with a full year ("YYYY" or "2023") or with an abbreviated year ("YY" or "23").
- Both a period (".") and a hyphen ("-") may be used to distinguish the day, month, and year in the date.
- Date can be written starting with either the day ("DD.MM.YYYY" or "14.06.2023") or the year ("YYYY.MM.DD" or "2023.06.14").
- The date and time can be separated by a space (" ") or its code ("%20"), or by a plus ("+").

Example of a request:

```
http://127.0.0.1:8080/command?type=setcell&login=root&password=&clientip=192.168.1.2&monitor=0&cell=4&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&mode=archive&starttime=14.09.2023+10:10:00&speed=2
```

Removing a channel from the grid cell

To delete a channel from one cell of the camera grid within the current screen profile, use the **clearcell** type of request.



The request lets changing grids only for those screen profiles that were created directly on the client device. Changing server profiles is not supported.




Any changes to the grid will be saved automatically within the current screen profile. A new profile will not be created.

Additional request parameters:

Parameter	Default value	Description
-----------	---------------	-------------

clientip	–	IP address of the device running the Eocortex client application. Necessary parameter
monitor	–	Number of the monitor in the current configuration of the client application. Starts from 0. Necessary parameter
cell	–	Cell number. Starts from 0. Necessary parameter



 The **clientip** value must contain the actual address of the **Eocortex** client. Using localhost (127.0.0.1) will cause an error even when accessing a client installed directly on the server.

Example of a request:

```
http://127.0.0.1:8080/command?type=cleargrid&login=root&password=&clientip=192.168.1.2&monitor=0&cell=4
```


Clearing the entire grid

To remove all used channels from the cells of the camera grid within the current screen profile, use the **cleargrid** type of request.

-  The request lets changing grids only for those screen profiles that were created directly on the client device. Changing server profiles is not supported.
-  Any changes to the grid will be saved automatically within the current screen profile. A new profile will not be created.

Additional request parameters:

Parameter	Default value	Description
clientip	–	IP address of the device running the Eocortex client application. Necessary parameter
monitor	–	Number of the monitor in the current configuration of the client application. Starts from 0. Necessary parameter

 The **clientip** value must contain the actual address of the **Eocortex** client. Using localhost (127.0.0.1) will cause an error even when accessing a client installed directly on the server.

Example of a request:

```
http://127.0.0.1:8080/command?type=cleargrid&login=root&password=&clientip=192.168.1.2&monitor=0
```

Setting channel to the guard mode

 Available in Eocortex version 2.1 and later

To change the state of the guard mode for a channel, use the **setguardian** type of request.

Additional request parameters:

Parameter	Default value	Description
clientip	–	IP address of the device running the Eocortex client application. Necessary parameter
monitor	–	Number of the monitor in the current configuration of the client application. Starts from 0. Necessary parameter

channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter
isguardmodeenabled	–	Switching the state of the armed mode. Possible values: true — activate the guard mode, false — deactivate. Necessary parameter

Example of a request:

```
http://127.0.0.1:8080/command?type=setguardian&login=root&password=&clientip=192.168.1.2&monitor=0&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&isguardianmodeenabled=true
```

Sending audio to the camera

 Available in Eocortex version 2.1 and later

Eocortex supports the ability to send audio streams for playback through the camera speaker. For this purpose, use POST-request to the **sendsound** resource. The URL of the request specifies the target parameters, the body of the request contains the audio data.



This request is primarily designed for use in third-party applications that act as **clients**. Additional libraries may be required to execute this request.



A request consists of a URL and a request body sent in JSON format. For successful execution it is necessary that the sender application knows how to handle such requests.

Additional URL parameters:

Parameter	Default value	Description
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter
clientid	–	Unique identifier of the transmission session



The value of the **clientid** parameter must be the **GUID** generated by the application that acts as the **client**.

Example of URL:

```
http://127.0.0.1:8080/sendsound?login=root&password=&channelid=66abc0c4-d4b7-4d71-8ed1-e7beadf0dc46&clientid=66abc0c4-d4b7-4d71-8ed1-e7beadf0dc46
```

In addition to the URL parameters listed above, the POST request must also contain the **ContentType = "multipart/form-data;"** header to specify the type of data transmitted in the body of the request.

Additional Information:

- The transmitted data must meet certain requirements. It is recommended to use third-party libraries (for example, NAudio: <https://github.com/naudio/NAudio>) to form audio data portion (audio coding), setting the following parameters during coding: Samplesrate=8000; Bitspersample=16; Number of channels = 1.
- The request is not permanently active. Data transfer starts only after the server receives a corresponding request from the client application.
- The request is not continuous. Within one request from the server, one portion of audio data is transmitted to the camera, after which the request will be considered as fulfilled.
- A transmission session can consist of several requests. A transmission session is a series of requests from a specific client device to a specific camera. It is allowed to use the same **clientid** within one transmission session to reduce server load.
- - Each transmission session must have a unique **clientid**. **GUID** generation for the **clientid** must be carried out by the application acting as the **client**.

Generating an event from an external system



Available in Eocortex version 2.1 and later

To create an event in the system log with the Event from external system type, use the **generateexternalevent** type of request.

Additional request parameters:

Parameter	Default value	Description
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter
systemname	–	Name of the external system
information	–	String with information about the event
eventcode	–	Numerical code of the event




This request is primarily designed to provide the option of basic integration with any third-party system not supported by **Eocortex**, including your own products. The values of the **systemname**, **information**, and **eventcode** parameters can be any.

Example of a request:

```
http://127.0.0.1:8080/command?type=generateexternalevent&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&systemname=testsystem&information=alarm&eventcode=5
```

In the **Event Log** of the **Eocortex Client** application, this event will look as follows:

Events		
Time	Type	Event description
21.10.2022 15:07:53		Camera 15. External event from the system testsystem. Code of event 5. Information: alarm.
<div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;">Time: 21 October 2022, 15:07:53.137 Camera: Camera 15. Type: Alarm. Event: External event. Initiator: System. Description: Camera 15. External event from the system testsystem. Code of event 5. Information: alarm.</div>		

Also, these events can be assigned to trigger actions in tasks (via the **Eocortex Configurator** application).

HTTP interface for operating PTZ features

In addition to the ability to control **Eocortex** components, the HTTP interface also lets you receive information and send control commands to PTZ devices connected to the system.

Such requests in general have the following format:

```
{Protocol}://{Server}:{Port}/ptz?command={Type}&login={Login}&password={Password}&channelid={Channel}&{Parameter}={Parameter value}
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	–	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol. Default ports: 8080 for http ; 18080 for https
Type	–	Type of request to the resource, if needed
Login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Channel	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request
Parameter	–	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	–	Value of the applied additional parameter



In comparison to the **Eocortex** component control commands, the commands listed below are sent via server to the PTZ device. The possibility and accuracy of their execution depends on the device and its integration in the **Eocortex**.

For requests containing commands to control the PTZ capabilities of the device, the server returns the single-type response about the results of the attempt to transmit the request to the device. If the request was successfully transmitted, the response body will contain only the **OK** status. Otherwise the response body will contain information about a transmission error.

Getting information about PTZ capabilities of the device

{...} Receiving response in JSON format is supported.

To get information about the PTZ control options supported by the system, use the **getcapabilities** type of request.

Additional request parameters:

Parameter	Default value	Description
responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/ptz?command=getcapabilities&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/ptz?command=getcapabilities&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&login=root&password=&responsetype=json
```

As a response, the server will send a list of PTZ features supported by the system, specifying through the parameter values those features that are supported by the device itself.

Example of a response in **XML** format:

```
<CameraCapabilities>
  <PtzCapabilities>
    <HomePositionSupports>true</HomePositionSupports>
    <MoveToSupports>true</MoveToSupports>
    <AreaZoomSupports>true</AreaZoomSupports>
    <ZoomSupports>true</ZoomSupports>
    <ContiniousZoomSupports>true</ContiniousZoomSupports>
    <AbsoluteZoomSupports>>false</AbsoluteZoomSupports>
    <MaxMinZoomSupports>>false</MaxMinZoomSupports>
    <AutoFocusSupports>true</AutoFocusSupports>
    <ManualFocusSupports>true</ManualFocusSupports>
    <ContiniousFocusSupports>true</ContiniousFocusSupports>
    <AutoIrisSupports>>false</AutoIrisSupports>
    <ManualIrisSupports>>false</ManualIrisSupports>
    <InfraredLightSupports>true</InfraredLightSupports>
    <WipersControlSupports>true</WipersControlSupports>
    <WasherControlSupports>true</WasherControlSupports>
    <SupportedStepMoveDirections>None</SupportedStepMoveDirections>
    <SupportedContiniousMoveDirections>Any</SupportedContiniousMoveDirections>
    <SetRelativePositionSupports>>false</SetRelativePositionSupports>
    <SetAbsolutePositionSupports>>false</SetAbsolutePositionSupports>
    <GetAbsolutePositionSupports>>false</GetAbsolutePositionSupports>
    <BlackWhiteModeSupports>>false</BlackWhiteModeSupports>
```

```

    <NightModeSupports>false</NightModeSupports>
  </PtzCapabilities>
  <Resolution>
    <Width>1920</Width>
    <Height>1080</Height>
  </Resolution>
</CameraCapabilities>

```

Example of a response in **JSON** format:

```

{
  "PtzCapabilities": {
    "HomePositionSupports": true,
    "MoveToSupports": true,
    "AreaZoomSupports": true,
    "ZoomSupports": true,
    "ContiniousZoomSupports": true,
    "AbsoluteZoomSupports": false,
    "MaxMinZoomSupports": false,
    "AutoFocusSupports": true,
    "ManualFocusSupports": true,
    "ContiniousFocusSupports": true,
    "AutoIrisSupports": false,
    "ManualIrisSupports": false,
    "InfraredLightSupports": true,
    "WipersControlSupports": true,
    "WasherControlSupports": true,
    "SupportedStepMoveDirections": "0",
    "SupportedContiniousMoveDirections": "255",
    "SetRelativePositionSupports": false,
    "SetAbsolutePositionSupports": false,
    "GetAbsolutePositionSupports": false,
    "BlackWhiteModeSupports": false,
    "NightModeSupports": false
  },
  "Resolution": {
    "Width": 1920,
    "Height": 1080
  }
}

```

Where:

Parameter	Description
HomePositionSupports	Support for returning the camera to the home position
MoveToSupports	Support for centering (moving the camera) on a set point
AreaZoomSupports	Support for zooming in on the selected frame area
ZoomSupports	Support for "step-by-step" (one-time) zoom level change
ContiniousZoomSupports	Support for "continuous" (performed until forced termination) zoom level change
AbsoluteZoomSupports	Support for absolute zoom
MaxMinZoomSupports	Support for switching to maximum or minimum zoom level

AutoFocusSupports	Support for automatic focus
ManualFocusSupports	Support for manual focus control
ContinuousFocusSupports	Support for "tracking" focus
ManualIrisSupports	Support for manual iris control
InfraredLightSupports	Support for infrared illumination control
WipersControlSupports	Support for mechanical wiper control
WasherControlSupports	Support for washer control
SupportedStepMoveDirections	A mask describing the available directions for "step-by-step" movement
SupportedContinuousMoveDirections	A mask describing the available directions for "continuous" movement
SetRelativePositionSupports	Support for setting the position relative to the current position of the device
SetAbsolutePositionSupports	Support for the ability to set the absolute position of the device
GetAbsolutePositionSupports	Support for the ability to get information about the current absolute position of the camera
BlackWhiteModeSupports	Support for switching the device to monochrome mode
NightModeSupports	Support for switching the device to "night" mode
Resolution	Current resolution of the main stream frame, where: <ul style="list-style-type: none"> • Width – frame width. • Height – frame height



For most PTZ capabilities there are only two possible values: true, which means that the capability is supported, and false, which means the opposite. The exceptions are the **SupportedStepMoveDirections** and **SupportedContinuousMoveDirections** parameters, whose value takes the numeric expression.

Getting device presets



Receiving response in JSON format is supported.

To get a list of preset positions of the camera, use the **getpresets** type of request.

Additional request parameters:

Parameter	Default value	Description
-----------	---------------	-------------

responsetype	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml, json
--------------	-----	--

Example of a request for data in **XML** format:

```
http://127.0.0.1:8080/ptz?command=getpresets&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b
```

Example of a request for data in **JSON** format:

```
http://127.0.0.1:8080/ptz?command=getpresets&login=root&password=&responsetype=json&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b
```

As a response, the server will return the list of all presets as they were created on the device.

Example of a response in **XML** format:

```
<ArrayOfString>
  <string>Door</string>
  <string>Lawn</string>
  <string>Window, 1st floor</string>
  <string>Window, 2nd floor</string>
  <string>Gate</string>
</ArrayOfString>
```

Example of a response in **JSON** format:

```
[
  "Door",
  "Lawn",
  "Window, 1st floor",
  "Window, 2nd floor",
  "Gate"
]
```



Eocortex receives the preset list as it is stored on the device itself. Changes to the list due to adding, changing or deleting presets depend on the device itself. In some cases, adding a new preset may change the order numbers of existing presets.

Setting a preset

To set the device to one of the preset positions, use the **gotopreset** type of request.

Additional request parameters:

Parameter	Default value	Description
index	–	The serial number of the preset in the list. Starts from 1. Necessary parameter

Example of a request:

```
http://127.0.0.1:8080/ptz?command=gotopreset&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&index=1
```

Getting a list of tours

To get a list of preconfigured tours on the camera, the **gettours** request type is used.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=gettours&channelid=154ecba1-bac2-4183-9e19-d1df6ea88ca2&responsetype=json
```

Example of a response in **JSON** format:

```
[
  {
    "Id": "52ae94-dcad-43be-aa19-5458f251136c",
    "Name": "Новый тур 1",
    "ContinueAfterUserPTZ": false,
    "ContinueType": "FromPreviousPreset",
    "ContinueAfterTime": 30,
    "States": [
      {
        "PresetNum": 1,
        "StayingInterval": 10
      },
      {
        "PresetNum": 2,
        "StayingInterval": 10
      },
      {
        "PresetNum": 3,
        "StayingInterval": 10
      }
    ]
  }
]
```



Responses to requests are available in **JSON** and **XML** formats.

Infrared lighting control

To send a command to the device to change the state (on/off) of the infrared lighting, the **switchinfraredlight** request type is used.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=switchinfraredlight&channelid=c1768c6c-1c01-4105-8384-c7cea4e53c30
```

Wiper control

To send a command to the device to change the state (on/off) of the wiper, the **switchwiper** request type is used.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=switchwiper&channelid=c1768c6c-1c01-4105-8384-c7cea4e53c30
```

Washer control

To send a command to the device to change the state (on/off) of the washer, the **runwasher** request type is used.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=runwasher&channelid=c1768c6c-1c01-4105-8384-c7cea4e53c30
```

"Continuous" movement

To send a command to the device for continuous movement in a specified direction, use the **startmove** type of request.

Additional request parameters:

Parameter	Default value	Description
panspeed	–	Speed of movement along the horizontal axis from -100 to 100. Necessary parameter
tiltspeed	–	Speed of movement along the vertical axis from -100 to 100. Necessary parameter
stoptimeout	500	Time in milliseconds after which the command will be terminated



The parameters **panspeed** and **tiltspeed** must be both specified in the request even if movement is required only on one of the axes. In such case the idling parameter must take 0 as the value.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=startmove&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&panspeed=2&tiltspeed=2&stoptimeout=100
```

"Continuous" change of focus

To send a command to the device to continuously change the focus, use the **startchange focus** type of request.

Additional request parameters:

Parameter	Default value	Description
speed	–	Speed of focus change from -100 to 100. Necessary parameter
stoptimeout	500	Time in milliseconds after which the command will be terminated

Example of a request:

```
http://127.0.0.1:8080/ptz?command=startchange focus&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&speed=5&stoptimeout=100
```

"Continuous" zoom

To send a command to the device to continuously change the degree of image magnification, use the **startzoom** type of request.

Additional request parameters:

Parameter	Default value	Description
speed	–	Speed of zoom change from -100 to 100. Necessary parameter
stoptimeout	500	Time in milliseconds after which the command will be terminated

Example of a request:

```
http://127.0.0.1:8080/ptz?command=startzoom&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&speed=2&stoptimeout=100
```

Termination of "continuous" actions

To terminate camera actions triggered by "continuous" commands, use the **stop** type of request.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=stop&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b
```

Automatic focus

To activate the automatic focus mechanism, use the **setautofocus** type of request.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=setautofocus&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b
```

Centering

To center the camera on a certain point of the frame, use the **moveto** type of request.

Additional request parameters:

Parameter	Default value	Description
width	–	Frame width in pixels. Necessary parameter
height	–	Frame height in pixels. Necessary parameter
x	–	The position of the centering point along the horizontal axis. Can take a value from 0 to the width parameter value, from left to right. Necessary parameter
y	–	The position of the centering point along the vertical axis. Can take a value from 0 to the height parameter value, from top to bottom. Necessary parameter

Example of a request:

```
http://127.0.0.1:8080/ptz?command=moveto&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&width=1920&height=1080&x=300&y=650
```



To get information about the width and height of the frame, use the [Getting information about PTZ capabilities](#) request.

"Step-by-step" movement

To send a command to the device for stepping in a specified direction relative to the current position, use the **move** request type.

Additional request parameters:

Parameter	Default value	Description
panstep	–	Step on the horizontal axis from -100 to 100. Necessary parameter
tiltstep	–	Step on the vertical axis from -100 to 100. Necessary parameter



The parameters **panstep** and **tiltstep** must be both specified in the request even if movement is required only on one of the axes. In such case the idling parameter must take 0 as the value.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=move&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&tiltstep=15&panstep=0
```

"Step-by-step" zoom

To send a command to the device to perform a step change in the magnification of the image, use the **zoom** type of request.

Additional request parameters:

Parameter	Default value	Description
zoomstep	–	Step from -100 to 100. Necessary parameter

Example of a request:

```
http://127.0.0.1:8080/ptz?command=zoom&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&zoomstep=10
```

Zooming the selected area (AreaZoom)

To zoom in on a certain part of the frame, use the showrect type of request.



Position of the area to be zoomed in is specified using a rectangle, defined by the parameters passed in the request: the coordinates of the upper left corner of the rectangle, as well as rectangle width and height relative to the specified position.

Additional request parameters:

Parameter	Default value	Description
frameWidth	–	Frame width in pixels. Necessary parameter
frameHeight	–	Frame height in pixels. Necessary parameter
x	–	Position of the upper left corner of the rectangle along the horizontal axis. Can take a value from 0 to the frameWidth parameter value, from left to right. Necessary parameter
y	–	Position of the upper left corner of the rectangle along the vertical axis. Can take a value from 0 to the frameHeight parameter value, from top to bottom. Necessary parameter
width	–	Width of the zoom area starting from the x parameter value in pixels. Necessary parameter
height	–	Height of the zoom area starting from the y parameter value in pixels. Necessary parameter



To get information about the width and height of the frame, use the [Getting information about PTZ capabilities](#) request.

Example of a request:

```
http://127.0.0.1:8080/ptz?command=showrect&login=root&password=&channelid=20d9884f-ae8c-45d3-ac5a-505ec258f01b&frameWidth=1920&frameHeight=1080&x=250&y=800&width=100&height=100
```

HTTP interface for receiving media data

In addition to the ability to receive text information about the system and manage its components, the **Eocortex** HTTP interface lets you access such content as frames and video from connected cameras using CGI requests.

Such requests in general have the following format:

```
{Protocol}://{Server}:{Port}/{Resource}?login={Login}&password={Password}&channelid={Channel}&{Parameter}={Parameter value}
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	–	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol. Default ports: 8080 for http ; 18080 for https
Resource	–	URI of the server resource to which the request is addressed
Login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Channel	–	The number, name, or unique identifier of the channel. Can be obtained by running the Receiving system configuration request. Necessary parameter
Parameter	–	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	–	Value of the applied additional parameter

Receiving a single frame

To get separate frames from the server for a particular camera, use a request to the **site** resource.

Such requests in general have the following format:

```
{Protocol}://{Server}:{Port}/site?login={Login}&password={Password}&{Parameter}={Parameter value}
```

Where:

Parameter	Default value	Description
-----------	---------------	-------------

Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	–	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol . Default ports: 8080 for http ; 18080 for https
Login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Parameter	–	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	–	Value of the applied additional parameter



If the login or password is incorrect, a **401** (Unauthorized) error will be returned.


Additional request parameters:

Parameter	Default value	Description
withcontenttype	false	Parameter that determines whether to specify a header with the type of data transmitted in the response from the server or not. Valid values: true , false . Optional parameter
mode	realtime	Parameter that defines the stream mode for transmitting frames. Valid values: realtime , archive . If the mode=archive , the starttime parameter must also be set. Optional parameter
starttime	–	Parameter specifying the time in the archive for which the frame should be transmitted. This parameter consists of a combination of date and UTC time in the format dd.MM.yyyy HH:mm:ss or dd.MM.yyyy HH:mm:ss.fff . Necessary parameter if mode=archive
resolution x	64	Required width of the requested frame in pixels. Cannot exceed the actual width of the frame. Optional parameter
resolution y	36	Required height of the requested frame in pixels. Cannot exceed the actual frame height. Optional parameter
streamtype	Last available value of the list	Parameter specifying the stream to receive frames. Possible values: <ul style="list-style-type: none"> • Main — the Main stream; • Alternative — the Additional 1 stream; • SecondAlternative — the Additional 2 stream; • ThirdAlternative — the Additional 3 stream. Availability of these values depends on the channel settings

Example of a request without additional parameters:

```
http://127.0.0.1:8080/site?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f
```

As a response, the server will return a frame from the real-time stream for the specified camera.

 By default, the frame will be taken from the last stream in the list. For example, if the server receives four streams from the camera, the frame will be taken from the stream "Additional 3", which corresponds to the fourth stream of the camera.

Example 1 of a request with additional parameters:

```
http://127.0.0.1:8080/site?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&streamtype=SecondAlternative
```

As a response, the server will return a frame from the Additional 2 stream, if possible.


Example 2 of a request with additional parameters:

```
http://127.0.0.1:8080/site?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&withcontenttype=true&mode=archive&starttime=01.01.2023 00:00:01&resolutionx=640&resolutiony=480
```

As a response, the server will return the frame from the archive recorded on January 01, 2023 at 00:00:01 UTC, adding to the response a header with the type of data and changing the resolution to 640x480, if possible.

Receiving raw video

One of the options to receive a video stream from the **Eocortex** server is a request to the **video** resource, which returns in response the data received by the server from the camera without transcoding.

 This request is primarily designed for use in third-party applications that act as **clients**. Additional decoding libraries may be required to display the received data.

Additional request parameters:

Parameter	Default value	Description
channel	–	Name of the channel within the current configuration. Can be obtained from the application interface and by running the Receiving system configuration request. Necessary parameter if channelid or channelnum is not used
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration . Necessary parameter if channel or channelnum is not used
channelnum	–	Number of the channel within the current configuration. Can be found out by manually counting the channels in the response to the Receiving system configuration request. Necessary parameter if channel or channelid is not used
sound	off	Parameter for receiving video and audio data of the channel within one connection. Possible values: on — enable audio reception, off — disable. If sound=on , server returns sound frames in G.711U format, interleaving them in data stream with video frames. Optional parameter
streamtype	Main	Parameter specifying the stream to receive frames. Possible values: <ul style="list-style-type: none">• Main — the Main stream;• Alternative — the Additional 1 stream;

		<ul style="list-style-type: none"> • SecondAlternative — the Additional 2 stream; • ThirdAlternative — the Additional 3 stream. Availability of these values depends on the channel settings
mode	realtime	Parameter that defines the stream mode for transmitting frames. Valid values: realtime , archive . If the mode=archive , the starttime parameter must also be set. Optional parameter
starttime	–	Parameter specifying the time in the archive for which frames should be transmitted. This parameter consists of a combination of date and UTC time in the format dd.MM.yyyy HH:mm:ss or dd.MM.yyyy HH:mm:ss.fff . Necessary parameter if mode=archive
speed	1	Parameter that defines the playback speed of the stream. Valid values: from 0.1 to 20 . Optional parameter, applies only when mode=archive



The **channel**, **channelnum** and **channelid** parameters are interchangeable, so the request should contain only one of them.

- The **channel** parameter passes the channel name as it is presented in the current configuration. If several channels with the same name exist in the configuration or if the channel name is changed in the configuration, collisions and errors may occur due to incorrect values of the parameter.
- The **channelnum** parameter passes as a value the serial number of the channel in the list of all channels in the current configuration. Using this parameter may cause difficulties when working with large systems. In addition, the channel number may change when channels were moved or deleted from the current configuration.
- The **channelid** parameter is a unique channel identifier (GUID) that is generated when the channel is created and remains unchanged throughout the lifetime of the channel in the configuration.

To avoid possible problems with requests, it is recommended to use the **channelid** parameter to specify the channel.



All examples of requests below are composed using the **channelid** parameter.

Example of a request without additional parameters:

```
http://127.0.0.1:8080/video?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f
```

In response to the request, the server returns an “endless” HTTP response containing video frames separated by headers.

Example of a response to a request:

```
HTTP/1.1 200 OK
...
Content-Type: multipart/x-mixed-replace; boundary=myboundary

-- myboundary
Content-Type: image/jpeg
Content-Length: 63125

<JPEG frame body>
```

If **sound=on**, video frames in the “endless” response are interleaved with audio frames.

Example of a request with a parameter **sound=on**:

```
http://127.0.0.1:8080/video?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&sound=on
```

Example of a response to a request with an audio frame:

```
HTTP/1.1 200 OK
...
Content-Type: multipart/x-mixed-replace; boundary=myboundary

-- myboundary
Content-Type: audio, PCMU
Content-Length: 1000

<G711U frame body>
```

To receive frames from the archive it is necessary to supplement the request with the appropriate parameters.

Example of a request for frames from the archive:

```
http://127.0.0.1:8080/video?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&mode=archive&startTime=01.01.2023 00:00:01&speed=3
```

In the “endless” response the server will start transmitting frames from the archive at x3 speed, starting from the archive frame taken on January 01, 2023 at 00:00:01 UTC. Response structure in this case is completely identical to responses with real-time frames.

Regardless of the request parameters applied, the response from the server will always contain the following parameters:

Parameter	Description
Content-Type	Header of the MIME type of the transmitted frame. Depending on the specified video format, it can take the following values: <ul style="list-style-type: none">• image/jpeg — video frame, MJPEG format.• video, mpeg4, I-frame — key frame, MPEG4 format.• video, mpeg4, P-frame — predicted frame, MPEG4 format.• video, h264, I-frame — key frame, H.264 format.• video, h264, P-frame — predicted frame, H.264 format.• audio, PCMU — audio frame, G.711U format.
Content-Length	Transmitted frame size header. The value is specified in bytes.



When transmitting reference frames of MPEG-4 and H.264 video formats, in addition to frame data itself, the initializing information for the decoder of the corresponding format will also be transmitted.

Receiving transcoded video in MJPEG format

When calling the [video](#) resource, the **Eocortex** server returns the video in the original format as it was received from the camera. For some applications and low-performance devices, decoding video in H.264 format or displaying MJPEG video in its original resolution can be a challenge.

For such cases, **Eocortex** is capable of sending video and audio streams transcoded by the mobile server. It is possible to receive transcoded streams using a request to the **mobile** resource.

Such requests in general have the following format:

```
{Protocol}://{Server}:{Port}/mobile?login={Login}&password={Password}&{Parameter}={Parameter value}
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	–	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol . Default ports: 8080 for http ; 18080 for https
Login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Parameter	–	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	–	Value of the applied additional parameter



If the login or password is incorrect, a **401** (Unauthorized) error will be returned.

Additional request parameters:

Parameter	Default value	Description
channel	–	Name of the channel within the current configuration. Can be obtained from the application interface and by running the Receiving system configuration request. Necessary parameter if channelid or channelnum is not used
channelid	–	Unique identifier of the channel. Can be obtained by running the Receiving system configuration . Necessary parameter if channel or channelnum is not used
channelnum	–	Number of the channel within the current configuration. Can be found out by manually counting the channels in the response to the Receiving system configuration request. Necessary parameter if channel or channelid is not used
sound	off	Parameter for receiving video and audio data of the channel within one connection. Possible values: on — enable audio reception, off — disable. If sound=on , server returns

		sound frames in G.711U format, interleaving them in data stream with video frames. Optional parameter
streamtype	Main	Parameter specifying the stream to receive frames. Possible values: <ul style="list-style-type: none"> • Main — the Main stream; • Alternative — the Additional 1 stream; • SecondAlternative — the Additional 2 stream; • ThirdAlternative — the Additional 3 stream. Availability of these values depends on the channel settings
soundformat	g711u	Specifies the format for transcoding an audio stream. Possible values: pcm , g711u , g711a , aac . Optional parameter
fps	Rate of the mobile server stream	Desired frame rate per second. The actual frame rate may differ from the requested one because it depends on many parameters. The maximum rate is limited by the settings of the transcoded stream for the mobile server. Optional parameter
oneframeonly	false	Parameter for single frame request with subsequent connection termination. Can be used as an alternative to site request. Possible values: true , false . Optional parameter
resolutionx	Low quality value	Desired frame width. Used to select the most suitable quality level for the transcoded stream. Optional parameter
resolutiony	Low quality value	Desired frame height. Used to select the most suitable quality level for the transcoded stream. Optional parameter
mode	realtime	Parameter that defines the stream mode for frames transmission. Possible values: realtime , archive . When applying the mode=archive parameter, it is necessary to set the starttime parameter as well. Optional parameter
starttime	–	Parameter specifying the time in the archive for which frames should be transmitted. This parameter consists of a combination of date and UTC time in the format dd.MM.yyyy HH:mm:ss or dd.MM.yyyy HH:mm:ss.fff . Necessary parameter if mode=archive
isforward	true	Direction of the archive playback. Possible values: true — the archive is played in chronological order, false — in reverse chronological order. Optional parameter, applies only when mode=archive
speed	1	Parameter that defines the playback speed of the stream. Valid values: from 0.1 to 20 . Optional parameter, applies only when mode=archive
withcontenttype	false	Parameter that determines whether to specify a header with the type of data transmitted in the response from the server or not. Valid values: true , false . Optional parameter



When using the **mobile** request, it is important to keep in mind that the term “stream” can refer to two entities at the same time:

- Original video stream
- Video stream transcoded by the mobile server

The original stream in this case is the stream received from the camera by the **Eocortex** server named **Main**, **Additional 1** (Alternative), **Additional 2** (SecondAlternative) or **Additional 3** (ThirdAlternative). Such stream is used as a source of frames for transcoding by the mobile server and is specified in the **streamtype** parameter

By transcoded stream is meant the same source stream, but converted by the mobile server in accordance with its own settings. This is what the server transmits in response to the request.



The **mobile** request uses the transcoding mechanisms of the mobile server to transmit streams. Therefore, the received stream is limited by the restrictions caused by the mobile server's settings.

The **mobile** server lets you set up to three quality levels of the stream to transcode: Low, Medium and High. For each quality level it is possible to specify its own frame resolution settings and their rate in the stream. For H.264, H.265, MPEG4 and MxPEG source streams it is also possible to set transcoding using only key frames.

When processing the **mobile** request, the server automatically chooses the level of quality closest to the specified values and applies the settings of this level to the stream to be transcoded: resolution of the level as the stream resolution, frame rate of the level as the maximum frame rate of the stream.

Example 1:

The request contains the following stream parameters: resolutionx=720, resolutiony=500, fps=20. The closest quality level to the specified parameters is **Medium** with frame resolution of 640x480 and fps=10. The resulting transcoded stream will be limited by the **Medium** quality level settings and will fully correspond to them.

Example 2:

The request contains the following stream parameters: resolutionx=600, resolutiony=460, fps=5. The closest quality level to the specified parameters is again **Medium** with the same settings. The resulting transcoded stream will then have a resolution 640x480 and fps=5.

To view the current mobile server settings, use the [Receiving system configuration](#) request – all the transcoding parameters are listed in the **MobileServerInfo** section.

To change the mobile server settings, use the [Mobile devices](#) tab of the server settings window in the **Eocortex Configurator** application.



The mobile server uses the same settings to send streams to mobile applications, to the Web Client, and in response to the mobile request. Changing mobile server settings can adversely affect other users of the system.

The mobile server has additional user access rights settings. The user sending the request must be granted with the **Connection via mobile devices and Web-Client** permission.

To check the current rights settings, you can use the [Receiving system configuration](#) request – the needed right is represented by the **CanGetTranscodedVideoFromMobileServer** parameter of the **UserGroup** section.

To change the rights settings, use the [Users](#) section of the **Eocortex Configurator** application.



The **channel**, **channelnum** and **channelid** parameters are interchangeable, so the request should contain only one of them.

- The **channel** parameter passes the channel name as it is presented in the current configuration. If several channels with the same name exist in the configuration or if the channel name is changed in the configuration, collisions and errors may occur due to incorrect values of the parameter.
- The **channelnum** parameter passes as a value the serial number of the channel in the list of all channels in the current configuration. Using this parameter may cause difficulties when working with large systems. In addition, the channel number may change when channels were moved or deleted from the current configuration.
- The **channelid** parameter is a unique channel identifier (GUID) that is generated when the channel is created and remains unchanged throughout the lifetime of the channel in the configuration.

To avoid possible problems with requests, it is recommended to use the **channelid** parameter to specify the channel.



All examples of requests below are composed using the **channelid** parameter.

Example of a request 1:

```
http://127.0.0.1:8080/mobile?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f
```

As a response, the server will start transmitting the stream according to the lowest quality level of the configured.

Example of a request 2:

```
http://127.0.0.1:8080/mobile?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&resolutiony=480&sound=on
```

As a response, the server will automatically determine the closest quality level according to its settings and start transmitting the stream according to its settings.

Example of a request 3:

```
http://127.0.0.1:8080/mobile?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&resolutiony=480&oneframeonly=true&mode=archive&starttime=01.01.2023 00:00:01
```

As a response, the server will automatically determine the closest quality level by the values of the settings and transmit a single frame from the archive, taken on January 01, 2023 at 00:00:01, and then terminate the transmission session.



If it is supposed to play the resulting transcoded stream in a browser, add the **withcontenttype=true** parameter to the request.

Receiving an archive fragment as MP4 video file

In addition to receiving streaming video and audio data, **Eocortex** also provide the option for exporting archive fragments as MP4 files. To do this, use requests to the **exportarchive** resource.



The function is supported only for cameras whose archive is recorded in H.264, H.265, or MJPEG formats.

Additional request parameters:

Parameter	Default value	Description
channelid	–	Unique identifier of the channel. Can be obtained by running the Obtaining system configuration request. Necessary parameter
fromtime	–	Date and time of the beginning of the period for which the information is requested. The time must be specified in the UTC time zone in DD.MM.YYYY hh:mm:ss format. Necessary parameter
totime	–	Date and time of the end of the period for which the information is requested. The time must be specified in the UTC time zone in DD.MM.YYYY hh:mm:ss format. Necessary parameter
sound	off	Parameter for adding audio data from the archive to the exported file. Available for Windows servers only. Possible values: on — export audio, off — do not export. Optional parameter
usetimestamps	false	Parameter that enables overlaying a timestamp with the recording time of the displayed frame on top of the video. Possible values: true — to overlay timestamp, false — to not overlay. Optional parameter
fromDevice	false	Parameter requesting to export the archive stored directly on the camera or DVR, if they support the function of accessing the archive. Possible values: true — export the archive from the device, false — export the archive from the server. Optional parameter
addhvc1tagforhvc	false	Option to add a header that allows playback of the exported fragment of the H.265 format on Apple devices. Slightly increases the duration of the export. Possible values: true — add header, false — do not add. Optional parameter

Example of a request:

```
http://127.0.0.1:8080/exportarchive?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&fromtime=01.01.2023 00:00:00&totime=01.01.2023 00:30:00&sound=on&usetimestamps=true
```

As a response to this request, the server will prepare an MP4 file containing a fragment of the archive with an audio track and timestamps, recorded on January 01, 2023 between 00:00:00 and 00:30:00. Downloading of the file will start automatically if the connection to the server is not interrupted by closing the application.



The request has several technical limitations that must be taken into account when using it:

- Requests for exporting a fragment of the archive are added to the server's queue for execution when received. The maximum queue size is 10 requests.
- The maximum number of simultaneously processed requests is 5. Thus, when the queue is fully loaded, only the first 5 requests will be processed, while the remaining 5 will be on hold.
- All requests above the maximum queue size will be discarded with message code 500.
- The maximum duration of the exported fragment is 1 hour.

RTSP interface for receiving video and sound

The RTSP interface is used for receiving video and audio by client applications using the RTSP protocol. This interface supports the H.264 codec and, optionally, MJPEG (MJPEG is disabled by default).



An alternative way of generating a URL to access the RTSP interface is described on the [Generation of RTSP links to cameras in Eocortex Configurator application](#) page of the Administrator's Guide.

To switch it on, it is required to launch **Eocortex Configurator**, and make sure that the checkbox **Accept RTSP connections** on the page of server settings in **Network** section is checked. The same tab also shows the RTSP port for making connections.

Server address: 192.168.101.223:8080

Archive Network Mobile devices Watchdog timer settings SSL certificate Synchronization with an external system Other

- Allow server location according to UPnP protocol
- Accept RTSP connections (to broadcast in H.264, H.265 and MJPEG)
- RTSP port (for TCP or HTTP connections): 554
- Allow Mjpeg broadcasting via RTSP ⓘ
- Accept ONVIF connections ⓘ
Username: onvif Password: ●●●●
- Enable integration with Unified Data Storage and Processing Center
- Allow multicasting
- Realtek PCIe GBE Family Controller (rt640x64)

Apply server settings Cancel

To make a RTSP connection it is possible to use TCP (RTSP over TCP) or HTTP (RTSP over HTTP) connections. UDP connections (RTSP over UDP) are not supported.



By default, MJPEG broadcasting via RTSP protocol is off, because this protocol supports only MJPEG frames coded in Baseline mode. Thus, transcoding is required for broadcasting video streams coded in other MJPEG modes, which in its turn will increase server load. Additionally, MJPEG transcoding may cause lower frame rate as compared with the frame rate of the camera.

Connection to the server is made by an RTSP client, for example, VLC, with the following connection string:

```
rtsp://{Server}:{Port}/rtsp?login={Login}&password={Password}&{Channel}={Channel value}&{Parameter}={Parameter value}
```

Where:

Parameter	Default value	Description
Server	–	Domain name or IP address of the Eocortex server
Port	554	RTSP protocol network port

Login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Channel	–	Channel specification method: channelid or channelnum
Channel value	–	Channel GUID for channelid or channel number for channelnum
Parameter	–	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	–	Value of the applied additional parameter



The **channelid** and **channelnum** parameters are interchangeable, so the request must contain only one of them. The **channelnum** parameter uses the channel serial number in the list of channels as a value. Any configuration change may cause the number to shift, which may cause errors in the request.

It is recommended to use a **channelid** that uses the unchangeable camera GUID that was generated when the channel was created.

RTSP request can also contain additional parameters that provide additional features:

Parameter	Default value	Description
sound	off	Parameter for receiving video and audio data of the channel within one connection. Possible values: on — enable sound reception, off — disable. Optional parameter
streamtype	Main	Parameter specifying the stream to receive frames. Possible values: <ul style="list-style-type: none"> • Main — the Main stream; • Alternative — the Additional 1 stream; • SecondAlternative — the Additional 2 stream; • ThirdAlternative — the Additional 3 stream. Availability of these values depends on the channel settings
mode	realtime	Parameter that defines the stream mode for transmitting frames. Valid values: realtime , archive . If the mode=archive , the starttime parameter must also be set. Optional parameter
starttime	–	Parameter specifying the time in the archive for which frames should be transmitted. This parameter consists of a combination of date and UTC time in the format dd.MM.yyyy HH:mm:ss or dd.MM.yyyy HH:mm:ss.fff . Necessary parameter if mode=archive
speed	1	Parameter that defines the playback speed of the stream. Valid values: from 0.1 to 20 . Optional parameter, applies only when mode=archive
isforward	true	Direction of the archive playback. Possible values: true — the archive is played in chronological order, false — in

		reverse chronological order. Optional parameter, applies only when mode=archive
--	--	--

Example of a request 1:

```
rtsp://127.0.0.1:554/rtsp?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f
```

As a response, the server will start transmitting video frames from the real-time stream for the specified channel.

Example of a request 2:

```
rtsp://127.0.0.1:554/rtsp?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&sound=on
```

As a response, the server will start transmitting video and audio frames from the real-time stream for the specified channel.

Example of a request 3:

```
rtsp://127.0.0.1:554/rtsp?login=root&password=&channelid=706c4691-3d90-41e3-8789-76eb9810648f&sound=on&mode=archive&starttime=01.01.2023 23:59:59&isforward=false&speed=2
```

As a response, the server will start transmitting video and audio frames from the archive of the specified channel for January 01, 2023, starting playback in reverse chronological order from 23:59:59 at the rate of x2.

HTTP interface for managing automatic switching of views

The HTTP interface make it possible to you to set an automatic switching profile and get information about it.

Example request:

```
{Protocol}://{Server}:{Port}/{Resource}?type={Type}&{Parameter}={Parameter value}&login={Login}&password={Password}
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	—	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol. Default ports: 8080 for http ; 18080 for https
Resource	—	URI of the server resource to which the request is addressed
Type	—	Request type, this type can take the following values: setautoswitchviews and getautoswitchviews
Login	—	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	—	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
Parameter	—	An additional parameter that specifies the request itself or the answer to it. Depending on the request, it may be possible to apply multiple additional parameters at the same time
Parameter value	—	Value of the applied additional parameter

Setting automatic switching profile

The **setautoswitchview** request type is used to set the automatic switching profile.

Additional request parameters:

Parameter	Default value	Description
clientip	—	IP address of the device on which the client application is running. Necessary parameter
monitor	—	Number of the monitor in the current configuration of the client application. Starts from 0. Necessary parameter
autoswitch viewid	—	A unique identifier created by the administrator in the Eocortex configurator, used to create an automatic switching view profile and apply it to the client application

Response example:

```
http://127.0.0.1:8080/command?type=setautoswitchview&clientip=127.0.0.1&monitor=0&autoswitchviewid=d50ce608-978e-4a49-b0f6-4406ae7f2975&login=root&password
```

Receiving automatic switching profiles

The **setautoswitchview** query type is used to set an automatic switching profile.

Additional request parameters:

Parameter	Default value	Description
response type	xml	Format of the returned data representation. If not specified in the request, the default value is used. Optional parameter. Possible values: xml , json

Response example:

```
http://127.0.0.1:8080/command?login=root&type=getautoswitchviews&password=&response type=json
```

Example of a response in JSON format:

```
[
  {
    "Id": "d50ce608-978e-4a49-b0f6-4406ae7f2975",
    "Name": " New automatic switching 1" },
  {
    "Id": "ae491b1a-8d3b-4031-a3fc-d84b94781ebe",
    "Name": " New automatic switching 2"}
]
```

Note: The **hostname** variable has been added to HTTP requests for executing commands in the **Eocortex** client, allowing to specify the client address by **hostname** and IP address. The **clientip** variable, similar in function to **hostname**, has been retained for backward compatibility. In case of simultaneous use of variables in a request, only the value of **hostname** will be used.

Eocortex API with XML interface



In order to work with the XML interface of the system, it is necessary to use applications that are able to send the XML-format body of the request to the specified URL.

XML interface provides the ability to send XML requests to the Eocortex server and receive data in the same format as a response. To send a request in XML format, use the following URL:

```
{Protocol}://{Server}:{Port}/xml
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	–	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol. Default ports: 8080 for http ; 18080 for https

In the body of the request specify the basic parameters using the following structure:

```
<?xml version="1.0" encoding="utf-8" ?>
<request>
  <server_login>root</server_login>
  <server_pass_hash></server_pass_hash>
  <request_name>get_people_counters</request_name>
  <request_params>
    ...
  </request_params>
</request>
```

Below is a description of the request body parameter assignment:

Parameter	Default value	Description
server_login	–	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
server_pass_hash	–	md5-hash of the Eocortex user password. If no password is specified for the user, this parameter can be left blank or not specified in the request
request_name	–	String name of request type
request_params	–	Container for setting the parameters specific to the request type defined in the request_name

In its turn, the server returns a response of the following format:

```
<?xml version="1.0" encoding="utf-8" ?>
<result>
  <request_name></request_name>
  <request_result>Ok</request_result>
  <request_msg>Request successful.</request_msg>
```



```
<request_time>20.09.2012 10:58:15</request_time>
<request_time_local>20.09.2012 16:58:15</request_time_local>
</result>
```

Where:

Parameter	Description
request_name	String name of request type
request_result	Request result: Ok — if the request was successful Error — if there were any errors
request_msg	String comment on the results of the request
request_time	Request time in UTC
request_time_local	Request time in local time zone



The response may also contain other parameters specific to the request type.

Receiving People Counting data

To get the data of the People Counting module, use the **get_people_counters** type of request.

This request has the following parameters:

Parameter	Default value	Description
channel_id	–	Unique identifier of the channel. Can be obtained by running the Obtaining system configuration request. Necessary parameter
search_time	–	The moment of time for which it is required to show counter data. The time is shown in yyyy-MM-dd HH:mm:ss format. GMT (UTC) time must be indicated.

Example of the request body:

```
<?xml version="1.0" encoding="utf-8" ?>
<request>
  <server_login>root</server_login>
  <server_pass_hash></server_pass_hash>
  <request_name>get_people_counters</request_name>
  <request_params>
    <channel_id>cacdd8e6-1c56-435c-86e3-6967d7494a50</channel_id>
    <search_time>2012-09-17 09:50:00</search_time>
  </request_params>
</request>
```

As a response, the server returns the following strings:

```
<in>434</in>  
<out>378</out>
```

Where:

Parameter	Description
in	Number of people who entered for the given counter
out	Number of people who exited for the given counter

Broadcasting video to a site

Video broadcast can be organized with the help of a mobile connections service of **Eocortex** server and the components on the client's side (in the browser).

Broadcasting via HTML5

 Available in Eocortex version 4.0 and later

Video broadcast to a site can be organized with the help of the mobile connections service of Eocortex server and an HTML5 player provided by the **Eocortex** Web Client.

To arrange a broadcast, it is necessary to generate a URL to the stream of the selected camera as follows:

```
{Protocol}://{Server}:{Port}/embedding/index.html#/embed?login={Login}&password={Password}&channelid={Channel}&channelstreamtype={Stream}&mode={Format}
```

Where:

Parameter	Default value	Description
Protocol	http	Network protocol selected for communication with the Eocortex server. The default is http , https availability is determined by the server settings
Server	—	Domain name or IP address of the Eocortex server
Port	8080	Network port according to the selected Protocol. Default ports: 8080 for http ; 18080 for https
Login	—	Name of the Eocortex user on whose behalf the request will be executed. The user must have access rights to the channels, functions and features of the system accessed as part of the request
Password	—	md5-hash of Eocortex user's password. If no password is specified for the user, the parameter value must be the md5-hash of the empty string (D41D8CD98F00B204E989800998ECF8427E)
Channel	—	Unique identifier of the channel. Can be obtained by running the Receiving system configuration request
Stream	—	Parameter specifying the stream to receive frames. Possible values: <ul style="list-style-type: none">• Main — the Main stream;• Alternative — the Additional 1 stream;• SecondAlternative — the Additional 2 stream;• ThirdAlternative — the Additional 3 stream. Availability of these values depends on the channel settings
Format	—	Preferred video format: MJPEG or H264



Only the **Login** parameter is case-sensitive, while the others can be specified in any case.

Example of URL:

```
https://188.17.220.37:18080/embedding/index.html#/embed?login=Website&password=5989aeb826edde08a1109deb5e61a4ba&channelid=d8112e29-fce9-40ea-bef4-a1c7b276ac98&channelstreamtype=Alternative&mode=H264
```

The video stream is broadcasted using **Eocortex Web Client** components, which requires the following settings to be checked in the **Eocortex Configurator** application before broadcasting:

- The stream must be allowed to be used by mobile applications and the Web Client. In the **Cameras** section, select the needed camera and make sure that the stream chosen for broadcasting has the **Mobile and web clients** option enabled.
- The user, on whose behalf the stream will be requested must have the right to use the embedded component. In the **Users** section, select the user group and click **Edit**. In the window that opens, select the **Basic** tab and make sure that the **Connection via mobile devices and Web-Client** permission is enabled.
- The user, on whose behalf the stream will be requested must have the right to view the camera selected for streaming. In the **Users** section, select the user group and click **Edit**. In the window that opens, select the **Cameras** tab and make sure that the **Surveillance** or **Surveillance and archive** permission is enabled for the selected camera.



To ensure the system security, it is recommended using a dedicated user for broadcasting video to the site, providing it with the minimum set of rights enough only for this purpose.

There are several ways to provide access to the stream using the generated link. For example, the link can be placed on the page as a hyperlink that opens a **separate tab with the player** or it can be used as content of the **iframe element** for playing directly on the page.

To open the player as a separate tab, place a hyperlink element (**<a>**) on the page, adding to it a previously generated link to the stream:

```
<a href="https://188.17.220.37:18080/embedding/index.html#/embed?login=Website&password=5989aeb826edde08a1109deb5e61a4ba&channelid=d8112e29-fce9-40ea-bef4-a1c7b276ac98&channelstreamtype=Alternative&mode=H264">Parking cam 1</a>
```



When opening the player as a separate tab, the width and height of the image are determined by the width and height of the opened tab, regardless of the original parameters of the received stream.

To place the player directly on the page, add the inline frame element (**<iframe>**) to the page code, adding to it the previously generated link to the stream:

```
<iframe src="https://188.17.220.37:18080/embedding/index.html#/embed?login=Website&password=5989aeb826edde08a1109deb5e61a4ba&channelid=d8112e29-fce9-40ea-bef4-a1c7b276ac98&channelstreamtype=Alternative&mode=H264" frameborder="0" width="{Width}" height="{Height}" allowfullscreen></iframe>
```



When embedding the player as the content of an iframe element, the image dimensions are determined by the specified **Width** and **Height** parameters, regardless of the original parameters of the received stream.

This method has some limitations that must be considered when using it:

- The method requires the browser to support the [Media Source API](#). Up-to-date versions of most browsers [support this API](#) by default, except **Safari for iOS**.
- Broadcasting is **only** available for **live video**. Access to the archive is not supported.
- Broadcasting is **only** available for **video**. Receiving audio from the camera is not supported in this method.
- Broadcasting is **only** available for **H264** and **MJPEG** streams. Other video formats are not supported.
- Broadcasting in **H264** is available only if the original stream from the camera is broadcasting in the **same format**. Transcoding of MJPEG streams to H264 format is not supported.
- Broadcasting in **MJPEG** is available both for cameras originally broadcasting **in this format** and for **H264 streams** by **transcoding** the original stream on the server.
- If any problems occur with displaying a stream in H264 format, **transcoding to MJPEG will be automatically applied** to the stream without the possibility of switching it back by the user.



Transcoding H264 to MJPEG consumes server resources and may cause an increased load on the server.

In the case of problems with broadcasting, make sure that:

- The specified **server**, **camera** and **stream** selected for broadcasting are **active** and **available** for use.
- The **user** has been **granted** with the necessary **permissions** to use Web Client components and receive the stream.
- All mandatory **parameters** of the link (credentials, camera ID, stream parameters) are **filled** out with **correct values**.



Detailed information on the reasons can be obtained in the Console of the browser. To do this, open the link to the stream as a separate tab, launch Developer Tools (F12) and switch to the Console tab.

Broadcasting via Flash (obsolete)



This method is obsolete due to the end of support for Flash technology by Adobe.

Video broadcast to a site can be organized with the help of a mobile connections service of Eocortex server and a Flash component on the client's side.

An example of a component used on the HTML page can be found in **Examples\SiteFlash** folder. It is necessary to indicate the parameters for connection to Eocortex server, as well as the required codec (H264 or MJPEG) and the identifier (name/number) of a channel from which the video will be broadcast, on HTML page (**index.html**).

Example of configuration:

```
var flashvars = {
    server: "demo.eocortex.com", // server address
    port: "8080", // server port
    login: "root", // user name
    password_hash: "", // md5 password hash
    mode: "MJPEG," // preferred video format
    channel: "1" // channel name, number or identifier
};
```

Identifiers of all channels in the system can be received with a specific request (see [Obtaining system configuration](#)).

Preferred video format (mode) parameter can be **MJPEG**, **H264**, or can be omitted. If preferred video format is not set, an appropriate format will be set automatically. **H264** value can be defined only for the cameras which broadcast H264-coded streams. **MJPEG** can be defined for all cameras, but it can cause increased server load if recoding from H.264 is required.

Broadcasting via JavaScript (obsolete)



This method is obsolete because it causes increased load of Eocortex server and provides inferior quality of broadcasting to site as compared with other options.

Video broadcast to a site can be organized with the help of Eocortex server and JavaScript component on the client's side. A script for the client's side and the example of its usage on the HTML page can be found in the **Examples\Site\frameReceiver.js** folder. In the script it is necessary to indicate the parameters for connection to Eocortex server, as well as the identifier (name/number) of a channel from which the video will be broadcast and the required size of the area to which video frames will be output.

Example of script configuration:

```
var serverUrl = "http://95.23.84.1:8080" /*server URL*/
var login = "root" /*user with rights to watch channel being broadcast*/
var password = ""; /*MD5 hash of user password in upper case or empty string in case of blank password*/
var channelnum = 0; /*ordinal number of channel in general configuration, counting from 0*/
var drawWidth = 577; /*display area width in pixels*/
var drawHeight = 432; /*display area height in pixels*/
```

An example of script using the channel identifier instead of its ordinal number can be found in **Examples\Site\frameReceiver_id.js** folder. There must be a tag `<imgname='frontImage' />` on the HTML page where MJPEG-coded video stream will be displayed.



It is not recommended to change display area size dynamically, because it will cause a substantial increase of resources used by mobile connections service since it transcodes initial video stream into MJPEG and then divides the received stream among many clients (sessions). The use of different resolutions will also cause additional load to the mobile connections service.