# Cinegy

Event Manager API Manual

# Table of Contents

**Cinegy Event Manager API** **1**

**IPlayOutExEventSink Interface** **1**

OnEvent 2

**IExEventHandler Interface** **2**

Init 3

SendEvent 3

ShutDown 4

**IEventHandlerInfoEx Interface** **4**

GetInfo 5

GetPluginName 5

**IEventHandlerSetup Interface** **5**

Setup 6

**Samples** **7**

# Cinegy Event Manager API

Cinegy Air has the possibility to add the controlling event to any item which will be sent to the event service. This service will process events in the specified way. Thus the system flexibility and scalability can be achieved and for each particular case the specified handler can be created.

The event itself is a text command with 5 parameters:

| Parameter | Type | Description |
|---|---|---|
| Device | String | Unique device name |
| Command | String | Device command |
| Parameter 1 | String | Context parameter 1 |
| Parameter 2 | String | Context parameter 2 |
| Parameter 3 | String | Context parameter 3 |

It's not recommended to use the following names for devices, they are reserved for the system use:

- LOGO;

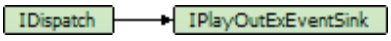
- RTP_IN.

Technically the event handler is an independent service which implements COM object with the following interface.

# IPlayOutExEventSink Interface

Interface provides method for events processing.

**Class Hierarchy**




**Syntax**

```
[ object, uuid(646BD034-19BD-415A-A47B-5F12C790062B), dual, nonextensible,
pointer_default(unique) ]
interface IPlayOutExEventSink : IDispatch;
```

**Methods**

| | Name | Description |
|---|---|---|
| | OnEvent | Method handles the event. |

# OnEvent

Method handles the event.

**Syntax**

```
[ id(1) ]
HRESULT OnEvent(
    [in] BSTR bstrDevice,
    [in] BSTR bstrCommand,
    [in] BSTR bstrOp1,
    [in] BSTR bstrOp2,
    [in] BSTR bstrOp3
);
```

**Parameters**

*bstrDevice*

specifies the device name.

*bstrCommand*

specifies the device command.

*bstrOp1*

specifies the context parameter 1.

*bstrOp2*

specifies the context parameter 2.

*bstrOp3*

specifies the context parameter 3.

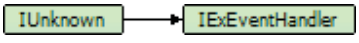
**Returns**

Returns S_OK if successful or an error value otherwise.

# IExEventHandler Interface

Base interface for plug-in initialization and events handling. This interface should be implemented by each handler.

**Class Hierarchy**



**Syntax**

```
[ object, uuid(BF89009C-AFC2-4e0d-BF5C-2FD4602C03E1), pointer_default(unique) ]
interface IExEventHandler : IUnknown;
```

**Methods**

| | Name | Description |
|---|---|---|
|  | Init | Method initializes the plug-in. |
|  | SendEvent | Method handles the event. |
|  | ShutDown | Method shuts down the plug-in. |

# Init

Method initializes the plug-in.

**Syntax**

```
HRESULT Init(
    [in] IUnknown* piEventLog
);
```

**Parameters**

*piEventLog*

pointer to the object implementing the IEventLog interface for outputting information to the Log window (e.g. the service console window).

**Returns**

Returns S_OK if successful or an error value otherwise.

# SendEvent

Method handles the event.

**Syntax**

```
HRESULT SendEvent(
    [in] BSTR bstrDevice,
    [in] BSTR bstrCommand,
    [in] BSTR bstrOp1,
    [in] BSTR bstrOp2,
    [in] BSTR bstrOp3
);
```

**Parameters**

*bstrDevice*

specifies the device name.

*bstrCommand*

specifies the device command.

*bstrOp1*

specifies the context parameter 1.

*bstrOp2*

specifies the context parameter 2.

*bstrOp3*

specifies the context parameter 3.

**Returns**

Returns S_OK if successful or S_FALSE if event wasn't recognized.

## ShutDown

Method shuts down the plug-in.

**Syntax**

```
HRESULT ShutDown();
```

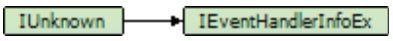
**Returns**

Returns S_OK if successful or an error value otherwise.

# IEventHandlerInfoEx Interface

Interface provides informational methods for passing the information about plug-in to the host-service.

**Class Hierarchy**




**Syntax**

```
[ object, uuid(535358E4-F1D1-4a58-9960-5B99782F16D3), pointer_default(unique) ]
interface IEventHandlerInfoEx : IUnknown;
```

**Methods**

| | Name | Description |
|---|---|---|
| | GetInfo | Method retrieves the plug-in information. |
| | GetPluginName | Method retrieves the plug-in name (to be displayed in the service setup utility). |

## GetInfo

Method retrieves the plug-in information.

**Syntax**

```
HRESULT GetInfo(
    [out] BSTR* pbstrDeviceInfo
);
```

**Parameters**

*pbstrDeviceInfo*

pointer to the variable that receives the information as a string.

**Returns**

Returns S_OK if successful or an error value otherwise.

## GetPluginName

Method retrieves the plug-in name (to be displayed in the service setup utility).

**Syntax**

```
HRESULT GetPluginName(
    [out] BSTR* pbstrName
);
```

**Parameters**

*pbstrName*

pointer to variable that receives the plug-in name.

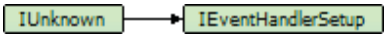
**Returns**

Returns S_OK if successful or an error value otherwise.

# IEventHandlerSetup Interface

Interface provides method for the plug-in setup.

**Class Hierarchy**




**Syntax**

```
[ object, uuid(E3BF96C9-F0F9-4bad-B6B0-D72F4D5F75C2), pointer_default(unique) ]
```

```
interface IEventHandlerSetup : IUnknown;
```

**Methods**

| | Name | Description |
|---|---|---|
|  | Setup | Method sets up the Event Handler. |

# Setup

Method sets up the Event Handler.

**Syntax**

```
HRESULT Setup(
    [in] HWND hwndParent
);
```

**Parameters**

*hwndParent*

handle to the dialog window with the setup parameters.

**Returns**

Returns S_OK if successful or an error value otherwise.

# Samples

In the PlayOutEventsSvc folder there is a sample of the event handler implementation (C++, ATL). It is registered as a service and implements the IPlayOutExEventSink interface. This sample outputs the event text and parameters to the console window.

In the EventServiceSample folder there are the sample implementation of the handler plug-in with the Event Service and the plug-in setup utility. Launch Register.bat from the "_install" directory. The plug-in project is in «SampleEventPlugin» directory. After building the project launch the setup utility EventsHandlerCfg.exe (from the «_install» directory) - the Sample Event Plugin will appear. Select it and tick off "Activate". After this all events will be sent to this plug-in.

EventSvcManager.exe is a main Events Service which receives events from the Air Service and transmits them to the event plug-ins.

In this way it is possible to add another handler plug-ins.

The project is built in Visual Studio 2015.