---

layout: doc title: Getting Started description: Learn how to install and configure Event Video Playback for your TwinCAT project permalink: /docs/getting-started/

Overview

Event Video Playback is a comprehensive solution for transforming TwinCAT Vision images into event-driven video recordings. This guide will walk you through the installation and initial setup process.

Prerequisites

Before you begin, ensure you have the following installed on your system:

For Engineering Development: - **Windows 10/11** - **TwinCAT Package Manager** - **TwinCAT 3.1 XAE** Build 4026 or higher - **TwinCAT Vision XAE** 5.8.4 or higher - **TwinCAT HMI XAE** 14.9 or higher

For Runtime Targets: - **Windows 10/11** - **TwinCAT Package Manager** - **TwinCAT 3.1 XAR** Build 4026 or higher - **TwinCAT Vision XAR** 5.8.4 or higher

Warning: Previous versions of the 4024 Tc_EventVideoPlayback legacy project must be uninstalled before using the new 4026 build.

TcPkg Package Signing Requirement

TwinCAT Package Manager only accepts officially signed packages from Beckhoff Automation GmbH & Co. KG by default. To use community packages, you need to temporarily disable signature verification. This applies to both locally hosted packages and remote packages; any 3rd party developed packages.

Security Notice: Disabling signature verification allows installation of third-party packages. Only use packages from trusted community sources. Review package contents and source code before installation. All community packages are provided "as is" without warranties.

Run this command in PowerShell as Administrator to disable signature checks:

```
tcpkg config unset -n VerifySignatures
```

Installation Methods

Choose the installation method that best fits your environment:

- **Online Installation** - For systems with internet access (recommended)
- **Offline Installation** - For air-gapped systems or manual installation

---

Online Package Feed Connection

Best for: Systems with internet connectivity and access to the Beckhoff USA Community package feed.

Add Package Feed via GUI

If you haven't already, add the Beckhoff USA Community package feed to TwinCAT Package Manager:

1. Open the TwinCAT Package Manager GUI
2. Click the Settings (Gear Icon) in the bottom left corner
3. Select Feeds

For the feed settings use:

```
https://packages.beckhoff-usa-community.com/stable/v3/index.json
```

For the feed name use:

```
Beckhoff USA Community Stable
```

Deselect the **Set credentials** option, as we do not need login for the feed. Select **Save** and agree to the disclaimer to be connected.

Add Package Feed via Powershell

```
tcpkg source add -n "Beckhoff USA Community Stable" -s "https://packages.beckhoff-usa-community.com/stable/v3/index.json"
```

Agree to the disclaimer to be connected.

Offline Package Configuration

Best for: Air-gapped systems, manual installations, or when you need a specific version.

Tip: Instead of downloading the packages from the releases section as noted below, you can also use the PowerShell command **tcpkg download [package name] -o [output location]**

Download from GitHub Releases

1. Navigate to the GitHub Releases page
2. Find the latest release (or the version you need)
3. Download the package file (`.zip`)
4. Transfer the files to your target system in an easy to remember location (Example: C:\Program Files\Beckhoff USA Community\Feeds\Local)

Add Local Package Feed via GUI

If you haven't already, add the Beckhoff USA Community package feed to TwinCAT Package Manager:

1. Open the TwinCAT Package Manager GUI
2. Click the Settings (Gear Icon) in the bottom left corner
3. Select Feeds

For the feed settings use:

```
C:\Program Files\Beckhoff USA Community\Feeds\Local
```

For the feed name use:

```
Beckhoff USA Community Local
```

Deselect the **Set credentials** option, as we do not need login for the feed. Select **Save**.

Add Package Feed via Powershell

```
tcpkg source add -n "Beckhoff USA Community Local" -s "C:\Program Files\Beckhoff USA Community\Feeds\Local"
```

Install Workloads

After the feed is added (locally or remote), and the VerifySignatures is disabled, you can now install the Workloads and Packages on the feed like you would normal Beckhoff Automation packages.

Next Steps

- Learn about PLC Library Usage for advanced features
- Configure Service Settings for your environment
- Add HMI Controls for video playback

Troubleshooting

Service Won't Start

- Verify .NET 8 Runtime is installed
- Check Windows Event Viewer for error messages
- Ensure no other service is using ADS port 26129

Videos Not Being Created

- Confirm TwinCAT Vision is saving images to the configured path
- Check service configuration file for correct paths
- Verify sufficient disk space is available

PLC Function Block Errors

- Ensure the library reference is added correctly
- Verify the service is running
- Check ADS communication settings

Support

Need help? Here are some resources:

- GitHub Issues - Report bugs or request features
- Beckhoff USA Community - Community support and discussions
- Documentation - Additional guides and references

---

layout: doc title: HMI NuGet Package Usage description: Integrate video playback controls into your TwinCAT HMI and WPF applications permalink: /docs/hmi-usage/

Overview

The Event Video Playback HMI NuGet package provides ready-to-use WPF controls for displaying and controlling video playback in your HMI applications. This guide covers installation, configuration, and usage.

Installation

Prerequisites

- **Visual Studio 2019 or later**
- **TwinCAT HMI Framework** (for HMI projects) or standalone **WPF application**
- **.NET 8 Framework**
- **NuGet Package Manager**

Install via NuGet

Option 1: Package Manager Console

```
Install-Package EventVideoPlayback.HMI
```

Option 2: NuGet Package Manager UI

1. Right-click on your project in Solution Explorer
2. Select **Manage NuGet Packages**
3. Click on **Browse** tab
4. Search for "EventVideoPlayback.HMI"
5. Click **Install**

Option 3: Beckhoff USA Community Package Feed

Add the Beckhoff USA Community package feed to your NuGet sources:

1. **Tools** > **Options** > **NuGet Package Manager** > **Package Sources**
2. Click the **+** button
3. Name: `Beckhoff USA Community`
4. Source: `https://packages.beckhoff-usa-community.com/nuget/v3/index.json`
5. Click **OK**

Then search for and install the package.

Video Player Control

Basic Usage

Add the video player control to your XAML:

```
<Window x:Class="YourNamespace.MainWindow"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:evp="clr-namespace:EventVideoPlayback.HMI.Controls;assembly=EventVideoPlayback.HMI"
       Title="Event Video Playback" Height="600" Width="800">

   <Grid>
       <evp:EventVideoPlayer x:Name="videoPlayer"
                             VideoSource="{Binding CurrentVideoPath}"
                             AutoPlay="True"
                             ShowControls="True"
                             Volume="0.5" />
   </Grid>
</Window>
```

Control Properties

Property	Type	Default	Description
`VideoSource`	string	null	Path to the video file to play
`AutoPlay`	bool	false	Automatically start playback when source is set
`ShowControls`	bool	true	Display playback controls
`Volume`	double	0.5	Volume level (0.0 to 1.0)
`IsMuted`	bool	false	Mute audio
`Loop`	bool	false	Loop video playback
`Stretch`	Stretch	Uniform	How video is stretched to fill control

Control Methods

```
// Play the video
videoPlayer.Play();
```

```
// Pause the video
videoPlayer.Pause();

// Stop the video
videoPlayer.Stop();

// Seek to specific position (in seconds)
videoPlayer.Seek(10.5);

// Load a new video
videoPlayer.LoadVideo("C:\\Videos\\MachineEvent_001.mp4");

// Take a snapshot
videoPlayer.SaveSnapshot("C:\\Snapshots\\snapshot.png");
```

Events

```
// Video loaded and ready to play
videoPlayer.VideoLoaded += (sender, e) =>
{
   Debug.WriteLine($"Video loaded: {e.VideoPath}");
};

// Playback started
videoPlayer.PlaybackStarted += (sender, e) =>
{
   Debug.WriteLine("Playback started");
};

// Playback paused
videoPlayer.PlaybackPaused += (sender, e) =>
{
   Debug.WriteLine("Playback paused");
};

// Playback completed
videoPlayer.PlaybackCompleted += (sender, e) =>
{
   Debug.WriteLine("Playback completed");
};

// Error occurred
videoPlayer.ErrorOccurred += (sender, e) =>
{
   MessageBox.Show($"Error: {e.ErrorMessage}", "Video Error");
};
```

Complete Example Application

XAML (MainWindow.xaml)

```
<Window x:Class="VideoPlaybackDemo.MainWindow"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:evp="clr-namespace:EventVideoPlayback.HMI.Controls;assembly=EventVideoPlayback.HMI"
       Title="Event Video Playback Demo"
       Height="700" Width="1000"
       Loaded="Window_Loaded">

   <Grid Background="#1a1a1a">
       <Grid.RowDefinitions>
           <RowDefinition Height="Auto"/>
           <RowDefinition Height="*"/>
           <RowDefinition Height="Auto"/>
       </Grid.RowDefinitions>

       <!-- Header -->
       <Border Grid.Row="0" Background="#D32F2F" Padding="15">
           <TextBlock Text="Event Video Playback Viewer"
                      FontSize="24"
                      FontWeight="Bold"
                      Foreground="White"/>
       </Border>

       <!-- Video Player -->
       <evp:EventVideoPlayer Grid.Row="1"
                             x:Name="videoPlayer"
                             Margin="10"
                             ShowControls="True"
                             Volume="0.7"/>

       <!-- Video List and Controls -->
       <Grid Grid.Row="2" Margin="10">
           <Grid.ColumnDefinitions>
               <ColumnDefinition Width="*"/>
               <ColumnDefinition Width="Auto"/>
           </Grid.ColumnDefinitions>

           <!-- Video List -->
           <ListBox Grid.Column="0"
                    x:Name="videoListBox"
                    SelectionChanged="VideoListBox_SelectionChanged"
                    Background="#2a2a2a"
                    Foreground="White"
                    Padding="5"
                    Height="150">
               <ListBox.ItemTemplate>
                   <DataTemplate>
                       <StackPanel Orientation="Horizontal">
                           <TextBlock Text="■" Margin="0,0,10,0"/>
                           <TextBlock Text="{Binding Name}"/>
```

```
                        </StackPanel>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>

            <!-- Control Buttons -->
            <StackPanel Grid.Column="1" Margin="10,0,0,0" VerticalAlignment="Center">
                <Button Content="Refresh List"
                        Click="RefreshList_Click"
                        Padding="15,5"
                        Margin="0,0,0,5"
                        Background="#D32F2F"
                        Foreground="White"
                        BorderThickness="0"
                        Cursor="Hand"/>
                <Button Content="Open Folder"
                        Click="OpenFolder_Click"
                        Padding="15,5"
                        Margin="0,0,0,5"
                        Background="#424242"
                        Foreground="White"
                        BorderThickness="0"
                        Cursor="Hand"/>
                <Button Content="Take Snapshot"
                        Click="TakeSnapshot_Click"
                        Padding="15,5"
                        Background="#424242"
                        Foreground="White"
                        BorderThickness="0"
                        Cursor="Hand"/>
            </StackPanel>
        </Grid>
    </Grid>
</Window>
```

Code-Behind (MainWindow.xaml.cs)

```
using System;
using System.IO;
using System.Linq;
using System.Windows;
using System.Diagnostics;
using EventVideoPlayback.HMI.Controls;

namespace VideoPlaybackDemo
{
    public partial class MainWindow : Window
    {
        private const string VIDEO_FOLDER = @"C:\TwinCAT\EventVideoPlayback\Videos";

        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            // Subscribe to video player events
            videoPlayer.VideoLoaded += VideoPlayer_VideoLoaded;
            videoPlayer.ErrorOccurred += VideoPlayer_ErrorOccurred;
            videoPlayer.PlaybackCompleted += VideoPlayer_PlaybackCompleted;

            // Load initial video list
            RefreshVideoList();
        }

        private void RefreshVideoList()
        {
            try
            {
                if (!Directory.Exists(VIDEO_FOLDER))
                {
                    MessageBox.Show($"Video folder not found: {VIDEO_FOLDER}",
                                    "Error",
                                    MessageBoxButton.OK,
                                    MessageBoxImage.Error);
                    return;
                }

                var videoFiles = Directory.GetFiles(VIDEO_FOLDER, "*.mp4")
                                          .Select(f => new FileInfo(f))
                                          .OrderByDescending(f => f.CreationTime)
                                          .ToList();

                videoListBox.ItemsSource = videoFiles;

                // Auto-select first video
                if (videoFiles.Any())
                {
                    videoListBox.SelectedIndex = 0;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error loading video list: {ex.Message}",
                                "Error",
                                MessageBoxButton.OK,
                                MessageBoxImage.Error);
            }
        }

        private void VideoListBox_SelectionChanged(object sender, System.Windows.Controls.SelectionChangedEventArgs e)
        {
            if (videoListBox.SelectedItem is FileInfo fileInfo)
            {
                videoPlayer.LoadVideo(fileInfo.FullName);
            }
        }

        private void RefreshList_Click(object sender, RoutedEventArgs e)
        {
            RefreshVideoList();
        }

        private void OpenFolder_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (Directory.Exists(VIDEO_FOLDER))
                {
                    Process.Start("explorer.exe", VIDEO_FOLDER);
                }
                else
                {
                    MessageBox.Show($"Folder not found: {VIDEO_FOLDER}",
                                    "Error",
                                    MessageBoxButton.OK,
                                    MessageBoxImage.Warning);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error opening folder: {ex.Message}",
                                "Error",
                                MessageBoxButton.OK,
                                MessageBoxImage.Error);
            }
        }

        private void TakeSnapshot_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                string snapshotPath = Path.Combine(
                    VIDEO_FOLDER,
                    $"Snapshot_{DateTime.Now:yyyyMMdd_HHmmss}.png"
                );

                videoPlayer.SaveSnapshot(snapshotPath);

                MessageBox.Show($"Snapshot saved to:\n{snapshotPath}",
                                "Snapshot Saved",
                                MessageBoxButton.OK,
                                MessageBoxImage.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error saving snapshot: {ex.Message}",
                                "Error",
                                MessageBoxButton.OK,
                                MessageBoxImage.Error);
            }
        }

        private void VideoPlayer_VideoLoaded(object sender, VideoLoadedEventArgs e)
        {
            // Optional: Update UI when video loads
            Title = $"Event Video Playback - {Path.GetFileName(e.VideoPath)}";
        }

        private void VideoPlayer_ErrorOccurred(object sender, VideoErrorEventArgs e)
        {
            MessageBox.Show($"Video playback error:\n{e.ErrorMessage}",
                            "Playback Error",
                            MessageBoxButton.OK,
                            MessageBoxImage.Error);
        }

        private void VideoPlayer_PlaybackCompleted(object sender, EventArgs e)
        {
            // Optional: Auto-play next video
            if (videoListBox.SelectedIndex < videoListBox.Items.Count - 1)
            {
                videoListBox.SelectedIndex++;
            }
        }
    }
}
```

Integration with TwinCAT HMI

For TwinCAT HMI projects, the control can be integrated with symbol bindings:

```
<evp:EventVideoPlayer VideoSource="{Binding PLC.VideoPath, Mode=OneWay}"
                     AutoPlay="{Binding PLC.AutoPlayEnabled, Mode=OneWay}"
                     Volume="{Binding HMI.VolumeLevel, Mode=TwoWay}"/>
```

Styling and Customization

Custom Control Template

You can customize the appearance of the video player:

```xml
<evp:EventVideoPlayer x:Name="videoPlayer">
   <evp:EventVideoPlayer.Style>
       <Style TargetType="evp:EventVideoPlayer">
           <Setter Property="Background" Value="#1a1a1a"/>
           <Setter Property="Foreground" Value="White"/>
           <Setter Property="BorderBrush" Value="#D32F2F"/>
           <Setter Property="BorderThickness" Value="2"/>
       </Style>
   </evp:EventVideoPlayer.Style>
</evp:EventVideoPlayer>
```

Performance Tips

1. **Preload Videos**: Load videos in background threads to avoid UI freezing
2. **Hardware Acceleration**: Ensure GPU acceleration is enabled
3. **File Formats**: Use H.264 (avc1) codec for best compatibility
4. **Resolution**: Match video resolution to display size
5. **Memory Management**: Dispose of video player when not in use

Troubleshooting

Video Won't Play

- Verify the video file exists and is accessible
- Check that the codec is supported (H.264 recommended)
- Ensure .NET 8 runtime is installed
- Try playing the video in Windows Media Player to verify it's valid

Control Not Visible

- Check that the NuGet package is correctly installed
- Verify XAML namespace declaration
- Ensure the control has proper Width/Height or is in a sized container

Performance Issues

- Reduce video resolution
- Use hardware acceleration
- Close other resource-intensive applications
- Check CPU and GPU usage

Best Practices

1. **Error Handling**: Always subscribe to ErrorOccurred event
2. **Resource Cleanup**: Dispose of video player when closing window
3. **File Validation**: Verify video files exist before loading
4. **User Feedback**: Show loading indicators for long operations
5. **Testing**: Test with various video formats and resolutions

Next Steps

- Review PLC Library Usage to trigger video recordings
- Configure Service Settings for optimal performance
- Check Getting Started for installation help

---

layout: doc title: PLC Library Usage description: Complete guide to using Event Video Playback function blocks in your TwinCAT PLC projects permalink: /docs/plc-usage/

Overview

The Event Video Playback PLC library provides easy-to-use function blocks for integrating event-driven video recording into your TwinCAT applications. This guide covers the complete API and usage patterns.

Library Reference

Adding the Library

In your TwinCAT PLC project:

1. Right-click on **References**
2. Select **Add Library**
3. Search for "EventVideoPlayback"
4. Select the library and click **OK**

The library will now be available for use in your PLC programs.

Core Function Blocks

FB_EventVideoRecorder

The main function block for recording event-driven videos from TwinCAT Vision images.

Declaration

```
VAR
    fbVideoRecorder : FB_EventVideoRecorder;
END_VAR
```

Input Parameters

Parameter	Type	Description
`sVideoName`	STRING	Unique name for the video file

`bExecute`	BOOL	Rising edge triggers video creation
`tPreEventTime`	TIME	Duration of pre-event video to include (default: 10s)
`tPostEventTime`	TIME	Duration of post-event video to include (default: 5s)
`sImagePath`	STRING	Path to TwinCAT Vision image folder
`bLogToEventLogger`	BOOL	Enable automatic Event Logger integration

Output Parameters

Parameter	Type	Description
`bDone`	BOOL	TRUE when video creation is complete
`bBusy`	BOOL	TRUE while video is being created
`bError`	BOOL	TRUE if an error occurred
`nErrorID`	UDINT	Error code (0 = no error)
`sVideoPath`	STRING	Full path to the created video file

Usage Example

```
PROGRAM MAIN
VAR
    fbVideoRecorder : FB_EventVideoRecorder;
    bMachineError   : BOOL;
    bEventTriggered : BOOL;
    sEventName      : STRING(255);
END_VAR

// Detect machine error event
IF bMachineError AND NOT bEventTriggered THEN
    bEventTriggered := TRUE;
    sEventName := CONCAT('MachineError_', TO_STRING(NOW()));
END_IF

// Record video on event
fbVideoRecorder(
    sVideoName := sEventName,
    bExecute := bEventTriggered,
    tPreEventTime := T#15S,  // 15 seconds before event
    tPostEventTime := T#10S,  // 10 seconds after event
    sImagePath := 'C:\TwinCAT\Vision\Images',
    bLogToEventLogger := TRUE
);

// Reset trigger when done
IF fbVideoRecorder.bDone OR fbVideoRecorder.bError THEN
    bEventTriggered := FALSE;
END_IF

// Handle errors
IF fbVideoRecorder.bError THEN
    // Log error code: fbVideoRecorder.nErrorID
    // Take corrective action
END_IF
```

Advanced Usage Patterns

Multiple Event Types

Handle different types of events with separate video recordings:

```
PROGRAM MAIN
VAR
    fbVideoQuality     : FB_EventVideoRecorder;
    fbVideoMaintenance : FB_EventVideoRecorder;
    fbVideoSafety      : FB_EventVideoRecorder;

    bQualityIssue      : BOOL;
    bMaintenanceNeeded : BOOL;
    bSafetyEvent       : BOOL;
END_VAR

// Quality issue recording
fbVideoQuality(
    sVideoName := 'QualityIssue_' + TO_STRING(NOW()),
    bExecute := bQualityIssue,
    tPreEventTime := T#20S,
    bLogToEventLogger := TRUE
);

// Maintenance event recording
```

```
fbVideoMaintenance(
    sVideoName := 'Maintenance_' + TO_STRING(NOW()),
    bExecute := bMaintenanceNeeded,
    tPreEventTime := T#30S,
    bLogToEventLogger := TRUE
);

// Safety event recording
fbVideoSafety(
    sVideoName := 'Safety_' + TO_STRING(NOW()),
    bExecute := bSafetyEvent,
    tPreEventTime := T#60S,
    tPostEventTime := T#30S,
    bLogToEventLogger := TRUE
);
```

Conditional Recording

Record videos only under specific conditions:

```
VAR
    fbVideoRecorder : FB_EventVideoRecorder;
    bEventOccurred  : BOOL;
    bRecordEnabled  : BOOL := TRUE;
    nEventSeverity  : UINT;
    nMinSeverity    : UINT := 5;  // Only record severity >= 5
END_VAR

// Only record if enabled and severity threshold met
IF bEventOccurred AND bRecordEnabled AND (nEventSeverity >= nMinSeverity) THEN
    fbVideoRecorder(
        sVideoName := CONCAT('Event_Severity_', TO_STRING(nEventSeverity)),
        bExecute := TRUE,
        bLogToEventLogger := TRUE
    );
END_IF
```

Sequence Recording

Record a sequence of related events:

```
TYPE E_RecordingState :
(
    IDLE,
    RECORDING,
    WAITING_FOR_NEXT,
    COMPLETE
);
END_TYPE
```

```
VAR
   fbVideoRecorder    : FB_EventVideoRecorder;
   eState             : E_RecordingState := IDLE;
   nSequenceCount     : UINT := 0;
   nMaxSequences      : UINT := 5;
   bStartSequence     : BOOL;
END_VAR

CASE eState OF
   IDLE:
       IF bStartSequence THEN
           nSequenceCount := 0;
           eState := RECORDING;
       END_IF

   RECORDING:
       fbVideoRecorder(
           sVideoName := CONCAT('Sequence_', TO_STRING(nSequenceCount)),
           bExecute := TRUE
       );

       IF fbVideoRecorder.bDone THEN
           nSequenceCount := nSequenceCount + 1;

           IF nSequenceCount >= nMaxSequences THEN
               eState := COMPLETE;
           ELSE
               eState := WAITING_FOR_NEXT;
           END_IF
       END_IF

   WAITING_FOR_NEXT:
       // Wait for condition to trigger next recording
       IF bTriggerNextRecording THEN
           eState := RECORDING;
       END_IF

   COMPLETE:
       // All sequences recorded
       eState := IDLE;
END_CASE
```

Error Handling

Error Codes

Error ID	Description	Resolution

0	No error	N/A
1001	Service not available	Check if EventVideoPlayback service is running
1002	Invalid image path	Verify sImagePath is correct
1003	Insufficient images	Ensure Vision system is capturing images
1004	Video creation failed	Check service logs for details
1005	Disk space insufficient	Free up disk space
1006	Invalid parameters	Check input parameter values

Error Handling Example

```
VAR
   fbVideoRecorder : FB_EventVideoRecorder;
   sErrorMessage   : STRING(255);
END_VAR

IF fbVideoRecorder.bError THEN
   CASE fbVideoRecorder.nErrorID OF
       1001:
           sErrorMessage := 'EventVideoPlayback service is not running';
           // Attempt to restart service or notify maintenance

       1002:
           sErrorMessage := 'Invalid image path configured';
           // Log configuration error

       1003:
           sErrorMessage := 'Not enough images available for video';
           // Check Vision system status

       1004:
           sErrorMessage := 'Video creation failed';
           // Check service logs and disk space

       1005:
           sErrorMessage := 'Insufficient disk space';
           // Trigger cleanup or notification

       ELSE:
           sErrorMessage := CONCAT('Unknown error: ', TO_STRING(fbVideoRecorder.nErrorID));
   END_CASE

   // Log error to Event Logger or HMI
END_IF
```

Best Practices

1. Unique Video Names

Always use unique video names to avoid conflicts:

```
// Good: Include timestamp
sVideoName := CONCAT('Event_', TO_STRING(NOW()));

// Better: Include machine ID and event type
sVideoName := CONCAT('Machine01_QualityIssue_', TO_STRING(NOW()));
```

2. Appropriate Time Windows

Choose pre/post event times based on your needs:

- **Fast events** (< 1 second): Use 5-10 seconds pre-event
- **Slow events** (> 10 seconds): Use 30-60 seconds pre-event
- **Post-event**: Usually 5-10 seconds is sufficient

3. Resource Management

Don't create videos too frequently:

```
VAR
   fbVideoRecorder    : FB_EventVideoRecorder;
   tLastRecording     : TIME;
   tMinInterval       : TIME := T#30S;  // Minimum 30s between recordings
END_VAR

IF (TIME() - tLastRecording) > tMinInterval THEN
   // Allow recording
   tLastRecording := TIME();
END_IF
```

4. Event Logger Integration

Enable Event Logger integration for automatic correlation:

```
fbVideoRecorder(
    sVideoName := sEventName,
    bExecute := bEvent,
    bLogToEventLogger := TRUE  // Automatically creates Event Logger entry
);
```

Performance Considerations

- **Image Buffer**: The service maintains a rolling buffer of images
- **CPU Usage**: Video encoding is performed by the service, not PLC
- **Network Impact**: Minimal - only ADS communication for commands
- **Disk I/O**: Configure video output path on fast storage

Next Steps

- Configure Service Settings for optimal performance
- Add HMI Controls for video playback
- Review Getting Started for installation help

---

layout: doc title: Service Configuration description: Configure the EventVideoPlayback Windows service for optimal performance permalink: /docs/service-config/

Overview

The EventVideoPlayback Service is a Windows background service that handles video creation and automatic file management. This guide covers all configuration options and troubleshooting steps.

Configuration File Location

The service configuration file is located at:

```
C:\Program Files\Beckhoff USA Community\EventVideoPlayback\Service\EventVideoPlaybackService.config.json
```

Open this file with Notepad, Visual Studio, or any text editor.

Configuration Parameters

Default Configuration

```
{
 "CodecFourCC": "avc1",
 "VideoDeleteTime": 1,
 "AdsPort": 26129,
 "MaxFolderSize": 250
}
```

CodecFourCC

Type: String **Default:** `"avc1"` **Description:** The video codec used to create MP4 videos from image sequences.

The `avc1` codec (H.264) is recommended as it provides: - Excellent compression - Wide compatibility with web browsers - Support for TwinCAT HMI playback - Good balance between quality and file size

Supported Codecs

The service supports multiple codecs, but not all are web-compatible. For TwinCAT HMI compatibility, stick with these recommended options:

Codec	FourCC	Description	Web Compatible	Recommended
H.264	`avc1`	Most common, best compatibility	✓	**Yes**
H.264	`avc3`	Alternative H.264 variant	✓	Yes
H.265/HEVC	`hev1`	Better compression, newer	✓	Maybe
H.265/HEVC	`hvc1`	Alternative HEVC variant	✓	Maybe
MPEG-4	`mp4v`	Older standard	✓	No
VP9	`vp09`	Google's codec	✓	No
AV1	`av01`	Newest, best compression	■	No

Changing the Codec

To change the codec, edit the configuration file:

```
{
 "CodecFourCC": "hev1",  // Changed to H.265
 "VideoDeleteTime": 1,
 "AdsPort": 26129,
 "MaxFolderSize": 250
}
```

■ **Important:** After changing any configuration, you must restart the service for changes to take effect.

VideoDeleteTime

Type: Decimal (floating-point) **Default:** `1` **Units:** Days **Description:** How long video files are kept before automatic deletion.

The service automatically cleans up old videos based on this setting. This helps manage disk space and keep only relevant videos.

Examples

```
// Keep videos for 1 day (default)
"VideoDeleteTime": 1

// Keep videos for 12 hours
"VideoDeleteTime": 0.5

// Keep videos for 1 week
"VideoDeleteTime": 7

// Keep videos for 30 days
"VideoDeleteTime": 30

// Keep videos for 2 hours
"VideoDeleteTime": 0.083
```

Calculation Reference

Duration	Value	Calculation
1 hour	0.042	1/24
6 hours	0.25	6/24
12 hours	0.5	12/24
1 day	1.0	24/24
3 days	3.0	-
1 week	7.0	-

1 month	30.0	-

AdsPort

Type: Integer **Default:** `26129` **Description:** The ADS port number the service listens on.

■ **WARNING: DO NOT CHANGE THIS VALUE**

The PLC function blocks are configured to communicate with the service on port 26129. Changing this value will break PLC communication.

If you have a specific need to use a different port, you must: 1. Change this configuration value 2. Recompile the PLC library with the new port number 3. Update all PLC projects using the library

MaxFolderSize

Type: Integer **Default:** `250` **Units:** Megabytes (MB) **Description:** Maximum total size of the video folder.

When a new video is created, the service checks the total folder size. If it exceeds this limit, the oldest videos are automatically deleted to free space.

Examples

```
// Limit to 250 MB (default)
"MaxFolderSize": 250

// Limit to 1 GB
"MaxFolderSize": 1024

// Limit to 5 GB
"MaxFolderSize": 5120

// Limit to 500 MB
"MaxFolderSize": 500
```

Size Planning

Consider these factors when setting folder size:

- **Video duration**: Longer pre/post event times = larger files
- **Image resolution**: Higher resolution = larger files
- **Frame rate**: More frames per second = larger files
- **Codec**: Different codecs have different compression ratios
- **Event frequency**: More frequent events = more videos

Example Calculation: - Average video duration: 30 seconds - Average file size: 5 MB per video - MaxFolderSize: 250 MB - Approximate capacity: ~50 videos

Applying Configuration Changes

After modifying the configuration file:

Option 1: Restart the Service

1. Open **Windows Services** (services.msc)
2. Find **EventVideoPlayback Service**
3. Right-click and select **Restart**

Option 2: Use Command Line

```
# Stop the service
net stop "EventVideoPlayback Service"

# Start the service
net start "EventVideoPlayback Service"
```

Option 3: Reboot

A system reboot will also restart the service with the new configuration.

Advanced Configuration

Example: High-Quality, Long Retention

For critical systems where video quality and retention are important:

```
{
 "CodecFourCC": "avc1",
 "VideoDeleteTime": 30,      // Keep for 30 days
 "AdsPort": 26129,
 "MaxFolderSize": 10240      // 10 GB
}
```

Example: Space-Constrained, Short Retention

For systems with limited disk space:

```
{
 "CodecFourCC": "avc1",
 "VideoDeleteTime": 0.5,     // Keep for 12 hours
 "AdsPort": 26129,
 "MaxFolderSize": 100        // 100 MB
}
```

Example: Maximum Compression

For maximum file size reduction:

```
{
 "CodecFourCC": "hev1",      // H.265 for better compression
 "VideoDeleteTime": 7,       // Keep for 1 week
 "AdsPort": 26129,
 "MaxFolderSize": 500        // 500 MB
}
```

Service Management

Checking Service Status

Windows Services GUI: 1. Press `Win + R`, type `services.msc`, press Enter 2. Find "EventVideoPlayback Service" 3. Check the Status column (should show "Running")

PowerShell:

```
Get-Service -Name "EventVideoPlayback*"
```

Command Prompt:

```
sc query "EventVideoPlayback Service"
```

Starting the Service

```
Start-Service -Name "EventVideoPlayback Service"
```

Stopping the Service

```
Stop-Service -Name "EventVideoPlayback Service"
```

Setting Startup Type

Automatic (recommended):

```
Set-Service -Name "EventVideoPlayback Service" -StartupType Automatic
```

Manual:

```
Set-Service -Name "EventVideoPlayback Service" -StartupType Manual
```

Troubleshooting

Service Won't Start

Check Event Viewer: 1. Open Event Viewer (eventvwr.msc) 2. Navigate to **Windows Logs** > **Application** 3. Look for errors from source "EventVideoPlayback"

Common Causes: - Missing .NET 8 Runtime - Invalid configuration file (JSON syntax error) - Port 26129 already in use - Insufficient permissions

Solutions:

```
# Verify .NET 8 Runtime is installed
dotnet --list-runtimes

# Check if port is in use
netstat -ano | findstr "26129"

# Run as administrator
net start "EventVideoPlayback Service"
```

Configuration Not Taking Effect

- Verify you saved the configuration file
- Ensure JSON syntax is valid (use a JSON validator)
- Restart the service after changes
- Check file permissions (service must be able to read the file)

Videos Not Being Deleted

- Verify `VideoDeleteTime` is set appropriately
- Check that the service has write permissions to the video folder
- Ensure the system clock is correct
- Review service logs for cleanup errors

Disk Space Issues

If you're running out of disk space:

Reduce MaxFolderSize: `json "MaxFolderSize": 100 // Reduce to 100 MB`

Reduce VideoDeleteTime: `json "VideoDeleteTime": 0.5 // Keep only 12 hours`

Manually clean old videos:

4. Navigate to the video output folder
5. Delete old MP4 files
6. Service will manage space going forward

Performance Issues

If video creation is slow:

- **Check CPU usage**: Video encoding is CPU-intensive
- **Use faster storage**: SSD is recommended for video output
- **Reduce image resolution**: Lower resolution = faster encoding
- **Consider codec**: H.264 (avc1) is generally fastest

Monitoring and Logging

Log File Location

Service logs are typically located at:

```
C:\Program Files\Beckhoff USA Community\EventVideoPlayback\Service\Logs\
```

Log Contents

Logs include: - Service startup and shutdown events - Video creation requests and completions - File cleanup operations - Errors and warnings - ADS communication status

Viewing Logs

Use any text editor or PowerShell:

```
# View latest log file
Get-Content "C:\Program Files\Beckhoff USA Community\EventVideoPlayback\Service\Logs\*.log" -Tail 50
```

Best Practices

1. **Regular Monitoring**: Check service status weekly
2. **Disk Space**: Ensure adequate free space (at least 2x MaxFolderSize)
3. **Backup Configuration**: Keep a copy of your configuration file
4. **Test Changes**: Test configuration changes in a development environment first
5. **Document Settings**: Document why you chose specific values

Next Steps

- Learn about PLC Library Usage
- Set up HMI Controls
- Review Getting Started