VisualAge Smalltalk

Multimedia Guide and Reference

Version 5.5

Note

Before using this document, read the general information under “Notices” on page vii.

August 2000

This edition applies to Version 5.5 of the VisualAge® Smalltalk products, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product. The term “VisualAge,” as used in this publication, refers to the VisualAge Smalltalk product set.

Portions of this book describe materials developed by Object Technology International Inc. of Ottawa, Ontario, Canada. Object Technology International Inc. is a subsidiary of the IBM® Corporation.

If you have comments about the product or this document, address them to: IBM Corporation, Attn: IBM Smalltalk Group, 621-107 Hutton Street, Raleigh, NC 27606-1490. You can fax comments to (919) 828-9633.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY, USA 10594.

IBM may change this publication, the product described herein, or both.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM	International Business Machines
OS/2®	VisualAge

The following terms are trademarks of other companies:

Microsoft®	Microsoft Corporation
Windows®	Microsoft Corporation

Windows is a trademark of Microsoft Corporation.

UNIX® is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

About this book

This book introduces you to the Multimedia feature of VisualAge Smalltalk. The feature is available for the OS/2 and Windows platforms.

What this book includes

This book has two parts:

- **"Part 1. Exploring multimedia development" on page 1** covers, step-by-step, how to build multimedia applications that run video, use MCI code, and otherwise use parts provided by the Multimedia feature.
- **"Part 2. Parts reference" on page 15** describes, in detai,l the parts and public interface available with the Multimedia feature.

Who this book is for

This book is for developers of applications who want to learn about the Multimedia feature. Though you do not have to be proficient at using VisualAge Smalltalk to complete the examples in this book, knowledge of how to build VisualAge applications is assumed.

About this product or feature

VisualAge enables you to quickly make client/server applications.

VisualAge makes you more productive by providing interactive visual programming tools and a set of parts that represent the client/server spectrum. You create applications by assembling and connecting parts. In many cases, you do not even have to write code.

If you do need to write code, VisualAge provides a state-of-the-art, integrated development environment. The programming language you use is Smalltalk, an industry-standard, highly productive, pure object-oriented language.

The Multimedia feature of VisualAge helps you become more productive by providing a set of parts that represent the multimedia spectrum. This feature allows you create your applications by assembling and connecting parts and, in many cases, without needing to write any code.

Multimedia is the combination of animation, sound, video, and other media into interactive computer applications. It is the interactive aspect of computer applications that makes multimedia different from other kinds of video and audio entertainment.

Using multimedia in your VisualAge application can enhance the quality of communication. Multimedia presents information in a richer, more sensory form that people are familiar with from their experiences with television, movies, and videos. Multimedia applications can convey information in a number of exciting ways that were not previously possible. The result can be reduced training time and greater personal productivity.

Conventions used in this book

This book uses several conventions that you might not have seen in other product manuals.

Tips and environment-specific information are flagged with icons:

 Shortcut techniques and other tips

 VisualAge for OS/2

 VisualAge for Windows

 VisualAge for UNIX platforms

These highlighting conventions are used in the text:

Highlight style	Used for	Example
Boldface	New terms the first time they are used	VisualAge uses **construction from parts** to develop software by assembling and connecting reusable components called **parts**.
	Items you can select, such as push buttons and menu choices	Select **Add Part** from the **Options** pull-down. Type the part's class and select **OK**.
Italics	Special emphasis	Do *not* save the image.
	Titles of publications	Refer to the *VisualAge Smalltalk User's Guide*.
	Text that the product displays	The status area displays *Category: Data Entry*.
	VisualAge programming objects, such as *attributes*, *actions*, *events*, *composite parts*, and *script names*	Connect the window's *aboutToOpenWidget* event to the *initializeWhereClause* script.

`Monospace font`	VisualAge scripts and other examples of Smalltalk code	`doSomething`
		`  | aNumber aString |`
		`  aNumber := 5 * 10.`
		`  aString := 'abc'.`

	Text you can enter	For the customer name, type `John Doe`

Tell us what you think

The VisualAge Smalltalk web page has a comment form. Please take a few moments to tell us what you think about this book. The only way for us to know if you are satisfied with our books or if we can improve their quality is through feedback from customers like you.

What's new in this version?

The Multimedia feature in Version 5.5 is like the previous version.

By default, the Multimedia feature uses the table-style properties views introduced in Version 4.0. If you prefer the notebook-style settings used in versions previous to Version 4.0, load the VA: VisualAge Custom Settings Views feature and specify the custom notebook-style settings in the Preferences window, which you open by selecting **Preferences** from the **Options** menu of the VisualAge Organizer.

Part 1. Exploring multimedia development

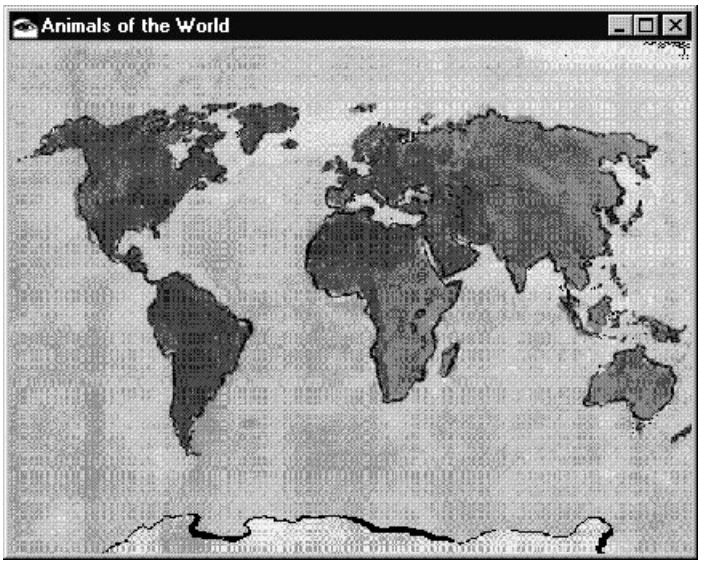
In this section, you learn how to build your first multimedia application. But there is so much more to learn about multimedia!

The remaining sections walk you through the multimedia component in more detail. An example is provided to show you how to build a more in-depth kiosk which helps explain the new authoring parts. Then you are shown how to add video playback to your applications with much more detail than is covered in your first application.

Chapter 1. Building your first multimedia application

This chapter guides you through building your first multimedia application with VisualAge. The steps you follow here teach you some of the basic principles that you will use in building other applications.

When finished, you will have an application that displays an image of the world.



Clicking on India plays a movie about one of the kinds of animals that live there:

The parts used to build this application are:
- A Form part to display the image of the world
- A Hotspot part on India
- A Digital Video Player to play a movie
- A Window to contain these parts

If you want to look at the finished application, load the **Multimedia Samples** feature and look at *MtMyFirstMultimediaView* in *AbtMultimediaSamplesApp*.

Creating an application that runs video

With the Multimedia feature loaded, create an application named, for example, *MyFirstMultimediaApp*. Next, create a visual part by selecting **New → Part** from the full **Parts** menu. Name the visual part *MyFirstMultimediaView*. After the Composition Editor displays, begin by adding parts to the application and changing some text.

Changing the text in the title bar

Instead of **Window** being the title of the main window, change it to *Animals of the World*.

Adding parts

Now, add three parts to the window.

Adding an image

To display an image of the world, do the following:

1. Select  , the Canvas category, and then select  , the Form part.
2. Drop the form on the **Animals of the World** window, double-click on it to open its settings, select the *backgroundGraphicsDescriptor* property and the **...** push button.
3. In the dialog that opens, select **Image**, specify `smworld.tif` with its fully-qualified path where the multimedia samples were installed, and then select **OK**.
4. So you can resize the image, select the *wallpaperStyle* property and, from its drop-down list, select *AbtScaledWallpaper*.
5. Close the properties window.

If you want the image to fill the entire space of a window, you do not need to add a form. A form and window each have wallpaper so if you want the `smworld.tif` to fill up the entire window, you can modify the wallpaper settings for the window.

Adding a hotspot

A movie showing an animal will play based on the country selected. To add an invisible push button, select  , the hotspot, from  , the Buttons category. Place it over India.

When the hotspot is drawn on the Composition Editor, it has a yellow border. This yellow border is not drawn when the application runs.

Adding a digital video player

The last part needed is the digital video player, which plays the appropriate animal movie. Select  , the digital video player, from  , the Multimedia category and place this nonvisual part on the free-form surface.

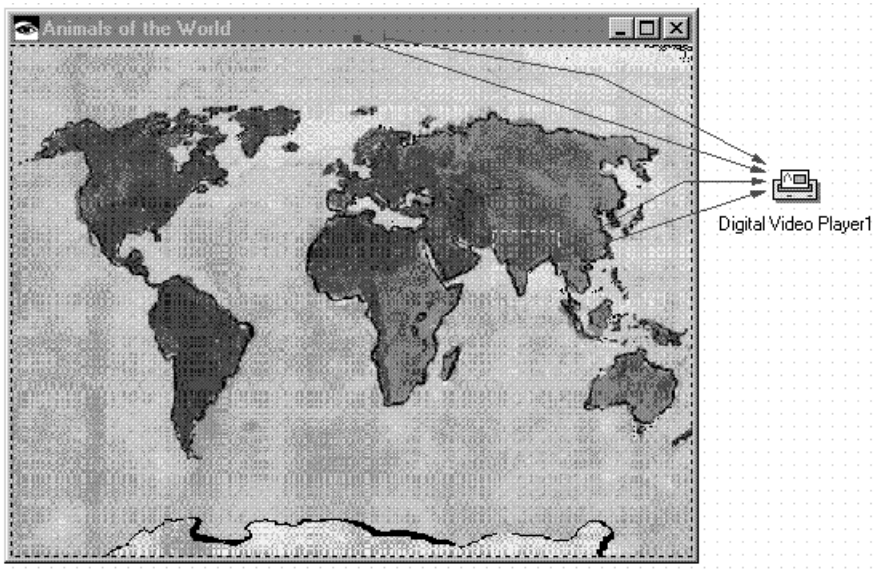
Connecting the parts

As you know, you add function to your application by making connections between the parts. To play a movie, you need to make a few connections to the digital video player:

1. To open the digital video player, connect the *aboutToOpenWidget* event of the window to the *open* action of the player.
2. To set what file to play, connect the *clicked* event of the hotspot to the *fileToLoad* attribute of the digital video player.

   Notice that you get a dotted line. A dotted line connection means that the connection is expecting a parameter that hasn't been set yet. To set the parameter, open up the connection's settings and click on the **Set parameters** push button. In the **value** entry field, type in `tiger.avi`. If the movie is not in the current VisualAge directory, be sure to include the path information.
3. To actually play the video, connect the *clicked* event of the hotspot to the *play* action of the digital video player.
4. The last thing that needs to be done is to close the player. To do this, connect the *aboutToCloseWidget* event of the window to the *close* action of the digital video player.

The free-form surface should now look as follows:



Finally, save and test your work.

On a 750 system in OS/2, you might find that running the video suspends processing.

Chapter 2. Adding video playback to an application

In this chapter, you learn how to add video playback to an application.

As you have seen in the previous chapters, you can create some powerful applications with just a few parts. One thing that hasn't been shown yet is one of the most common forms of media to date—**video**. "A picture is worth a thousands words" couldn't be more true in today's ever changing world.

The video support in VisualAge is only limited to whatever your operating system supports. If you are running OS/2 Warp, you can play AVI files recorded with different compressions, FLC and FLI animation files, and MPEG files.

Adding parts for playing video

The application you make in this chapter uses several parts that play video or that allow you to regulate the playing of video.

Begin by creating a new application. Add to it a visual part named *MyTahitiVideoView*. In the Composition Editor that opens, change the title of the Window part to `Tahiti video`.

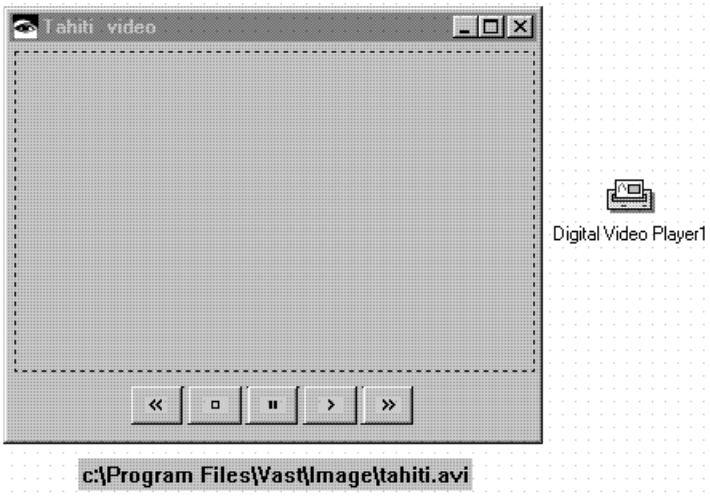
Select  , the digital video player, and place it on the free-form surface. This represents the device that plays digital video files.

To play and stop this video, add the motion buttons,  .

Since the video is to be displayed in the scene, add  , the video playback window to the scene. Open the video playback window's settings and set the *recomputeSize* property to *false* so the video scales to fit the window.

There are many different ways to specify what video file to play. One way would be to store it in a database along with the package name. Then, when a particular package for Tahiti is selected, you would know what video file to play. To simplify things, you are going to hardcode the video file using a label part. Select the label and add it to the nonvisual part of the free-form surface. Set its *object* property to `tahiti.avi`, including any path information if the video file is not in the current directory.

Your Composition Editor window should resemble:



Connecting the video pieces together

The connections you make are few and simple. There are two important connections that need to be made. First, connect the *aboutToOpenWidget* event of the window to the *openFile* action of the player. Next, connect the *aboutToCloseWidget* event of the window to the *close* action of the player. You need to open the player before you can do anything and you should always clean up when you are done with the player, so close it.

Once opened, a multimedia player must close before it, or another player, is opened again. Otherwise, you get a system hang or other unexpected results.

Telling the video player what file to play

Connect the *object* attribute of the label part to the *aFileName* attribute of the *aboutToOpenWidget-openFile* connection. The *openFile:* action immediate loads a video file. Thus, when the next action, like *play*, occurs in the player, the video file loads.

If you don't want to immediately load the file, use the *fileToLoad* attribute.

If you want the video to play in the video playback window, connect the *videoWindow* attribute of the video playback window to the *videoPlaybackWindow* attribute of the video player. If you do not want to play the video in a specific window, do not connect the *videoPlaybackWindow* attribute; the player creates a default desktop window and plays the video there.

When the video player closes, its attributes are set to *nil*. So, if you close the video player while a scene is still up and then reopen it, the video will not play in the video playback window. This is because VisualAge triggered the video playback window connection already and can't fire it again. A solution is to write a simple script that recreates this connection.

Telling the video player to play

So far, you have told the video player when to open, when to close, what to play and where to play it. But what about playing it?

Instead of making a number of connections to numerous buttons, you can use the motion buttons. Simply connect the *self* attribute of the video player to the *player* attribute of the motion buttons.

To make use of the intelligence built into the buttons, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This connection grays the appropriate buttons if a particular action cannot be performed. For instance, the rewind button grays if you are at the beginning of the file.

Now test your application and watch that video fly!

Chapter 3. Handling errors

The Last Error part provides you with error information for debugging or for providing message boxes. This enables you to debug your application as well as provide error messages to the users of your application.

Debugging multimedia applications

To help debug, instead of bringing up a message box every time there is an error, you can add fields to your views that display error information as soon as the error occurs. Whenever there is an error, these fields are updated and you don't have a message box to interact with. One way to do this is to tear off the *lastError* attribute and display its error values as strings in a label or text view.

Displaying messages

The easiest way for you to add error handling to your application is to tear off the *lastError* attribute of a part, such as the Digital Video Player, and connect the *errorOccurred* event of *lastError* to the *display* action of *lastError*. Whenever a multimedia parts error occurs, a message box is displayed to the user.

There are three different types of errors: critical, warning and information. You can control which errors are displayed in the message box by setting *errorThreshold*. There are three values that you can set:

1 Only critical errors
2 Both critical and warning errors
3 Critical, warning, and information errors

The default error threshold is 2. To change this, you can connect the *aboutToOpenWidget* event of your window to the *errorThreshold* attribute of *lastError*. Set the argument to be 1, 2, or 3.

Chapter 4. Writing your own MCI code

Intelligence and sophistication have been built into the media players. Down at the lowest level, this is all implemented by wrapping the Media Control Interface.

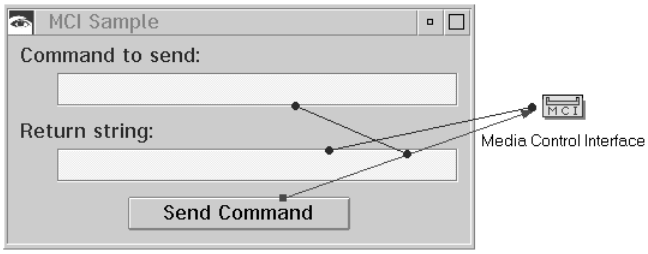
By using 

, the MCI part, you can write your own MCI code to do specialized tasks. Refer to your MCI toolkit information for the respective platforms to learn about MCI programming.

The MCI samples can be viewed in *MtMCIView* in *AbtMultimediaSamplesApp*.

Sending MCI commands in a view

You can use the MCI part in your views by making a few connections:



The *MtMCI* part does not exist on the parts palette so it must be added using the **Add Part** choice in the **Options** menu. When the push button is *clicked*, take the contents of the entry field and pass the contents (the command string) to the *send:* action of the MCI part. The operating system returns the *returnString* from the *mciSendString* call.

Of course, you can make this much more elaborate. For example, you might have a list that keeps track of all the strings that have been sent as well as their return values. Or you might want a list of pre-built messages to send one at a time.

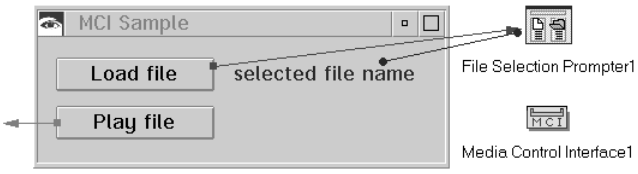
Sending MCI commands using a script

If you want to have a block of code run, just write a script. The following script opens the video player, loads a movie and plays it:

```
runMCI
   | mci |
   mci := (self subpartNamed: 'Media Control Interface').

   "Open the digital video device."
   mci send: 'open digitalvideo wait'.

   "Load the tahiti movie. It must be in the current directory.
    Or you must specify a full path name in the command."
   mci send: 'load digitalvideo tahiti.avi wait'.

   "Play the movie and don't return until the movie is done."
   mci send: 'play digitalvideo wait'.

   "Close the digital video device."
   mci send: 'close digitalvideo wait'.
```

Using the script with a view

You can use this code in combination with a view that gets user input such as requests to load and play a particular MCI file. For example, a simple view might have two push buttons, **Load file** and **Play file**, and use a file selection prompter for selecting an MCI file.



The *runMCI* script might read:

```
runMCI
  | mci file |
  mci := (self subpartNamed: 'Media Control Interface1').
  file := (self subpartNamed: 'File Selection Prompter1') selectedFileName.
  mci send: 'open digitalvideo wait'.
  mci send: 'load digitalvideo ',file,' wait'.
  mci send: 'play digitalvideo wait'.
  mci send: 'close digitalvideo wait'.
```

For the connections, you do the following:

- Connect the *clicked* event for **Load file** to the *prompt* action of the file selection prompter.
- Connect the *clicked* event for **Play file** to the *runMCI* script.
- Connect the *selectedFileName* attribute of the prompter to the *object* attribute of the label.

Part 2. Parts reference

This section provides descriptions of the attributes, events, and actions for multimedia parts. Where useful, it includes details on how you might use a part.

Chapter 5. Audio Wave Form File

The audio wave form file represents the actual medium that is being played by the audio wave form player.

To access the audio wave form file's public interface, you must first add an audio wave player part to your application. Then, you tear off the *currentFile* attribute from the player.

Part:

Class Name:
MtAudioWave

Attributes

Use the audio wave form file attributes when you want your application to reflect some information that is maintained in the audio wave form file. The following list describes the attributes available in the audio wave form file.

fileChanged (Boolean) - Read-only

The *fileChanged* attribute indicates whether the audio wave file has been modified as a result of a record operation.

For example, connect this to your user interface to show the user visually that the file has been modified like connecting to a check box.

fileName (String) - Read-only

The *fileName* attribute is the actual name of the audio or video file to be played.

For example, to display the name of a file in the title bar of a window, connect *fileName* to the *title* attribute of the window.

totalTime (String) - Read-only

The *totalTime* attribute is the total length of the media in hours, minutes, and seconds. The string is formated in the format of *HH:MM:SS*. If you want to display the elapsed time in a different format, use the time converter.

For example, to display the total length of the file, connect *totalTime* to the *label* attribute of a static text field. So, if the length was 1 hour, 32 minutes, and 25 seconds, the static text field would display *1:32:25*.

totalTimeAsMMTime (Integer) - Read-only

The *totalTimeAsMMTime* attribute is the total length of the media in MMTIME. One unit of MMTIME is equivalent to 1/3000 of a second, so for a file that was one hour in length (or 3600 seconds), the MMTIME value would be 10,800,000 (3600 seconds x 3000).

For example, to have a slider control reflect the current position of a wave file in OS/2, connect this attribute to the *maximumValue* attribute of the slider. See also the *location* attribute of the player.

Chapter 6. Audio Wave Player

Select the audio wave player part to play or record digital audio using files.

While **audio** refers to the sound waves that have a perceived effect on the human ear, **wave form** refers to the digital representation of the original audio sound wave.

Part:

Category:
Multimedia

Class Name:
MtAudioWavePlayer

Attributes

Use the Audio Wave Player attributes when you want your application to reflect some information that is maintained in the player. The following list contains the attributes that are available. Some of these attributes also have their own public interface. You can get to these interfaces by tearing off an attribute.

channel (Integer)

The *channel* attribute is the channel setting, indicating either mono or stereo recording. The default value is based on the audio hardware that is used.

For example, connect *channel* to the *contents* attribute of an edit field.

currentFile MtAudioWave) - Read-only

The *currentFile* attribute represents the file that will be played by the player. Typically, you do not make a connection to this attribute but instead, tear it off. By tearing the attribute off, you gain access to information about the specific file such as the file name and the file's length.

For example, if you want to display the name of the file that the Audio Wave Player is playing, tear off the *currentFile* attribute. You now have access to the current file's attributes and you can connect the *fileName* attribute to the *object* attribute of a label part.

elapsedTime (String) - Read-only

The *elapsedTime* attribute represents the time in hours, minutes, and seconds where the player is currently playing or positioned. The string is formatted in the format of *HH:MM:SS*. If you want to display the elapsed time in a different format, use the time converter.

For example, to display how much time has elapsed, connect *elapsedTime* to the *object* attribute of a label part.

fileToLoad (String)

The*fileToLoad* attribute represents the file against which the next operation will be performed. Use *fileToLoad* when you might be keeping the player open for an extended period of time. Instead of always doing a *close* and then an *openFile*, use *fileToLoad*. The next action that the player performs will compare the current file to the file to load. If they are different, then the player will load the new file and perform the requested action. The player must already be opened before *fileToLoad* will work. Also refer to performance improvements for additional information.

For example, to set this attribute, connect the *selectedItem* attribute of a list box to *fileToLoad*.

format (String)

The *format* attribute is the recording format, either Adaptive Differential Pulse Code Modulation (ADPCM) or Pulse Code Modulation (PCM).

For example, to display the current recording format, connect *format* to the *label* attribute of a static text field.

ADPCM is supported in OS/2 but not in Windows.

isMuted (Boolean)

The *isMuted* attribute indicates whether the sound is on or off for the player.

For example, connect *isMuted* to the *selection* attribute of the mute button to ensure that the button's appearance matches the state of the player.

isPaused (Boolean)

The *isPaused* attribute indicates whether play is suspended for the player.

For example, connect *isPaused* to the *selection* attribute of the pause button to ensure that the button's appearance matches the state of the player.

lastError (MtLastError) - Read-only

The *lastError* attribute represents the last multimedia error that occurred. Typically, you do not make a connection to this attribute but instead, tear it off. By tearing it off, you gain access to information about the specific error that occurred, such as the error number and related message text.

For example, if you want to display a message box when an error occurs, tear off the *lastError* attribute. You can now connect the *errorOccured* event to the *display* action of the last error object to display the message box.

location (Integer) - Read-only

The *location* attribute is the time in MMTIME where the player is currently playing or positioned. See also the *totalTimeAsMMTime* attribute of *currentFile*.

For example, to have a slider control reflect the current position of the video file in OS/2, connect this attribute to the *value* attribute of the slider. Or, in OS/2 or Windows, connect *location* to the *object* attribute of a label.

samplingPrecision (Integer)

The *samplingPrecision* attribute is the recording sampling precision, either 8 or 16 bits. The default value is based on the audio hardware that is used.

For example, connect *samplingPrecision* to the *contents* of an edit field to set it.

samplingRate (Integer)

The*samplingRate* attribute is the recording sampling rate, either 8, 11, 22, or 44 kHz. The default value is based on the audio hardware that is used.

For example, connect *samplingRate* to the *contents* of an edit field.

source (String)

The *source* attribute represents where the recording comes from, either line input or microphone.

For example, connect *source* to the *label* of a static text field.

Line in and microphone is supported in OS/2 but not in Windows.

volume (Integer)

The *volume* attribute represents the current volume setting for the player.

For example, connect *volume* to the *value* of a slider.

Volume is supported in OS/2 but not in Windows.

Actions

Use the Audio Wave Player actions when you want your application to tell the audio wave player to do something.

close

The *close* action closes the player and frees any associated resources. After the player closes, it must be opened again before it can play a file. If the player is currently performing some work, that work will stop.

For example, connect to *close* when you no longer need the player, such as when your view is about to close.

fastForward

The *fastForward* action rapidly moves the position of the media forward. This connection is not needed if you made the *player-self* and *stateChanged-setEmphasis* connections.

For example, connect to *fastForward* when the fast forward button is *clicked*.

mute

The *mute* action silences the player. This connection is not needed if you made the *player-self* connection.

For example, connect to *mute* when the **Mute** toggle button is *clicked*.

For the audio wave player, *mute* is supported in OS/2 but not in Windows.

open

The *open* action opens and initializes the player. Each player initializes with different default files:

- For the digital video player, the digital video file is `new.avi`.
- For the audio wave player, the audio wave file is `new.wav`.

For example, connect to *open* when you are initializing your window, such as when the *aboutToOpenWidget* event occurs.

openWaveFile: aFileName (String)

The *openWaveFile:* action opens and initializes the audio wave player with the audio wave file described by the *aFileName* parameter.

For example, connect to this when you are initializing your window, such as when the *aboutToOpenWidget* event occurs.

pause

The *pause* action suspends playing of the player but preserves the current position in the file. This connection is not needed if you made the *player-self* connection.

For example, connect to *pause* when the **Pause** toggle button is *clicked*.

play

The *play* action starts playing the player. This connection is not needed if you made the *player-self* connection.

For example, connect to *play* when the play button is *clicked*.

playShortWaveFile: fileName (String)

The *playShortWaveFile:* action opens the player, plays the file described by the *fileName* parameter, and then closes the player. The action will not return until the entire audio file is played. If it is a long file, this could give the appearance of a hung machine.

For example, this action is useful for playing short audio wave files.

record

The *record* action starts recording at the current location of the audio wave file. The record operation overwrites any information at the current location until the record operation stops.

For example, connect to *record* when the Record button is *clicked*. This connection is not needed if you made the *player-self* connection.

rewind

The *rewind* action rapidly moves the position of the media backward. This connection is not needed if you made the *player-self* connection.

For example, connect to *rewind* when the Rewind button is *clicked*.

saveFile

The *saveFile* action saves the changes that were made as a result of a record operation.

For example, you can write an event-to-script connection that checks to see if *fileIsChanged* is *true*. If it is *true*, then call this script to save your changes.

setChannelToMono

The *setChannelToMono* action sets the number of channels for recording to mono.

For example, connect to *setChannelToMono* when a radio button is *clicked*.

setChannelToStereo

The *setChannelToStereo* action sets the number of channels for recording to stereo.

For example, connect to *setChannelToStereo* when a radio button is *clicked*.

setFormatToADPCM

The *setFormatToADPCM* action sets the format type for recording to Adaptive Differential Pulse Code Modulation (ADPCM). ADPCM is a compressed form of Pulse Code Modulation. Using ADPCM reduces the amount of data stored, but at a loss of fidelity.

For example, connect to *setFormatToADPCM* when a radio button is *clicked*.

ADPCM is supported in OS/2 but not in Windows.

setFormatToPCM

The *setFormatToPCM* action sets the format type for recording to Pulse Code Modulation (PCM). Use PCM to achieve high fidelity recordings.

For example, connect to *setFormatToPCM* when a radio button is *clicked*.

setPrecisionTo16Bits

The *setPrecisionTo16Bits* action sets the sampling precision for recording to 16 bits. To achieve a high precision recording, use 16 bits; otherwise, use 8 bits. Using 16-bit precision increases the amount of data stored.

For example, connect to *setPrecisionTo16Bits* when a radio button is *clicked*.

setPrecisionTo8Bits

The *setPrecisionTo8Bits* action sets the sampling precision for recording to 8 bits. To achieve a high precision recording, use 16 bits; otherwise, use 8 bits. Using 8-bit precision reduces the amount of data stored.

For example, connect to *setPrecisionTo8Bits* when a radio button is *clicked*.

setSamplingRate: rate (Integer)

The *setSamplingRate:* action sets the sampling rate for recording with the rate described by the :hp1.rate:ehp1. parameter. The valid values are 8, 11, 22, or 44 kHz.

For example, connect to *setSamplingRate:* when a radio button is *clicked*.

setSourceLineIn

The *setSourceLineIn* action sets the connector type for recording to be line input.

For example, connect to *setSourceLineIn* when a radio button is *clicked*.

Line in is supported in OS/2 but not in Windows.

setSourceMicrophone

The *setSourceMicrophone* action sets the connector type for recording to be a microphone.

For example, connect to *setSourceMicrophone*when a radio button is *clicked*.

Microphone is supported in OS/2 but not in Windows. In Windows, microphone is the default and cannot be changed.

stop

The *stop* action halts the current action of the player. This connection is not needed if you made the *player-self* connection.

wrap

The *wrap* action toggles the wrap setting, which determines what the player does when the end or beginning of the media is reached. During play, when this attribute is set to *true*, the player continues play at the beginning of the media when the end is encountered. This connection is not needed if you made the *player-self* connection.

For example, connect to *wrap* when the **Wrap** toggle button is *clicked*.

Events

Use the audio wave player events when you want your application to be notified when a change has occurred in the audio wave player.

initialized

The *initialized*event occurs when the player finishes loading and querying the device for the attributes contained in the player.

For example, connect an action to this event to change from the wait pointer back to the arrow pointer.

majorErrorOccurred

The *majorErrorOccurred* event occurs when the player cannot fully initialize. One reason might be that the multimedia dynamic link library could not be found and loaded. Another reason might be that the player could not be found. You can connect an action to this event to inform the user that the player cannot initialize.

majorErrorOccurred is different from the *errorOccurred* event in the *lastError* object and does not generate error text. They are distinct so you can easily distinguish between a fatal error and a non-fatal error without having to write any code.

For example, connect to this event if you want to close your application or display a message to the user.

noVolumeRange

The *noVolumeRange* event occurs when the player does not support the full range of volume. Only 0 (off) and 100 (on) are supported.

For example, you can disable the volume slider with this event.

Volume is supported in OS/2 but not in Windows.

playWrapped

The *playWrapped* event occurs when the end of the media is encountered during a play operation and playing continued at the start of the media. Play will wrap only if *wrap* is set to *true*.

For example, you could connect to this event to update a slider which shows current position in a digital video or audio wave file.

reachedBegin

The *reachedBegin* event occurs when the beginning of the media is reached during play or other movement of the player.

For example, you can connect to this event to disable any buttons that are not available at the beginning of a digital video or audio wave file.

reachedEnd

The *reachedEnd* event occurs when the end of the media is reached during play or other movement of the player.

For example, you can connect to this event to disable any buttons that are not available at the end of a digital video or audio wave file.

stateChanged

The *stateChanged* event occurs when the state of the player changes. For example, the state goes from STOPPED to PLAYING when the play button is pressed.

For example, you could connect the *setEmphasis* action to this event.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

Opening and closing the application

When your application opens, you should make a connection to open the audio wave player. The easiest way to open the audio wave player is by connecting the *openedWidget* event of your window to the *openFile* action of the audio wave player. Then, connect the name of the file you want to open to the *aFileName* parameter of the *openFile* connection.

When your application is finished working with the audio wave player, we recommend that you close it. Then, when your application ends, the player does not remain open. Closing the audio wave player ensures that other applications that use audio will have access to the player. One example is to connect the *aboutToCloseWidget* event of your window to the *close* action of the audio wave player.

It is important that, once opened, a multimedia player close before it, or another player, opens again. Otherwise, the system might hang or another unexpected result might occur.

Error handling

The easiest way for you to add error handling to your application is to tear off the *lastError* attribute and connect the *errorOccured* event of *lastError* to the *display* action of *lastError*. Now, whenever an error occurs, a message box displays.

There are three different types of errors: critical, warning and information. You can control which errors are displayed in the message box by setting *errorThreshold*. There are three values that you can set:

1 Displays only critical errors.
2 Displays both critical and warning errors.
3 Displays critical, warning and information errors.

The default error threshold is 2. To change this, you can connect the *aboutToOpenWidget* event of your window to the *errorThreshold* attribute of *lastError*. Set the argument to be either 1, 2, or 3.

Performance improvement

Depending on how your application's user interface is designed, you might want to take advantage of the *fileToLoad* attribute. For example, you can build a user interface that contains both a list box which lists the audio wave form files and the motion buttons. To play the selected file, you have two alternatives. The first is

that every time you make a selection from the list, you open up the file. When the audio wave player receives an open, it loads up the file and prepares for playback. For large files, this can be time consuming. A better alternative is to update the *fileToLoad* attribute as the user maneuvers through the list. The file loads only when the user presses one of the buttons, such as **Play**.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a *playPressed-play* connection.

Playing a short file

If you use the *playShortWaveFile* action, please note the following:

- The system volume control must be used to control the sound.
- The elapsed time does not update until the play ends.
- If the digital video player is already open, calling this action closes the player.
- A short wave file (four seconds or less) cannot be stopped or paused. This is true whether you use *playShortWaveFile* or *play*.

Recording

This section applies only to the Windows version of VisualAge.

You cannot change the recording settings of an existing file. To record a file with different settings other than the default, you must open a new file with a file name of `new`. To do this, either connect to the *open* action, which implicitly opens a new file, or connect to the *openFile:* action and pass `new` as the file name.

Chapter 7. Digital Video File

The digital video file represents the actual medium that is being played by the digital video player.

To access the digital video file's public interface, you must first add a digital video player part to your application. Then, you tear off the *currentFile* attribute from the player.

Part:

Class Name:
MtDigitalVideo

Attributes

Use the digital video file attributes when you want your application to reflect some information that is maintained in the digital video file. The following list contains the attributes that are available in the digital video file.

fileName (String) - Read-only

The *fileName* attribute is the actual name of the audio or video file to be played.

For example, to display the title of your video file, `c:\myvideo.avi`, in the title bar of a window, connect *fileName* to *title* attribute of the window.

totalTime (String) - Read-only

The *totalTime* attribute is the total length of the media in hours, minutes, and seconds. The string is formatted in the format of *HH:MM:SS*. If you want to display the elapsed time in a different format, use the time converter.

For example, to display the total length of the file, connect *totalTime* to the *label* attribute of a static text field. So, if the length was 1 hour, 32 minutes, and 25 seconds, the static text field would display *1:32:25*.

totalTimeAsMMTime (Integer) - Read-only

The *totalTimeAsMMTime* attribute is the total length of the media in MMTIME. One unit of MMTIME is equivalent to 1/3000 of a second, so for a file that was one hour in length (or 3600 seconds), the MMTIME value would be 10,800,000 (3600 seconds x 3000).

For example, to have a slider control reflect the current position of a wave file in OS/2, connect this attribute to the *maximumValue* attribute of the slider. See also the *location* attribute of the player.

Chapter 8. Digital Video Player

Select the digital video player part to play digital video using files.

These files can contain both video and audio data or just video data. The video can either be played in a separate window on the desktop or imbedded in an existing window like a dialog.

Digital video can come in different formats. Look at your operating system multimedia documentation to see what multimedia input/output (MMIO) procedures are supported. For example, OS/2 Warp supports:

Audio/Video Interleaved (AVI)
This is referred to as software motion video because no specific hardware is required to play the video.

MPEG
This is video that requires hardware to be played. For example, you can use the ReelMagic adapter to play MPEG files.

FLC and FLI
These files are animation files.

Part: 

Category:
Multimedia

Class Name:
MtDigitalVideoPlayer

Attributes

Use the Digital Video Player attributes when you want your application to reflect some information that is maintained in the player. The following list contains the attributes that are available. Some of these attributes also have their own public interface. You can get to these interfaces by tearing off an attribute.

currentFile (MTDigitalVideo) - Read-only

The *currentFile* attribute represents the file inside the player. Typically, you do not make a connection to this attribute but, instead, tear it off. By tearing it off, you gain access to information about the specific file like the file name and its length.

For example, if you want to display the name of the file that the Digital Video Player is playing, tear off the *currentFile* attribute. You now have access to the current file's attributes and can connect the *fileName* attribute to the *object* attribute of a label.

elapsedTime (String) - Read-only

The *elapsedTime* attribute represents the time in hours, minutes, and seconds where the player is currently playing or positioned. The string is formatted in the format of *HH:MM:SS*. If you want to display the elapsed time in a different format, use the time converter.

For example, to display how much time has elapsed, connect *elapsedTime* to the *object* attribute of a label part.

fileToLoad (String)

The*fileToLoad* attribute represents the file against which the next operation will be performed. Use *fileToLoad* when you might be keeping the player open for an extended period of time. Instead of always doing a *close* and then an *openFile*, use *fileToLoad*. The next action that the player performs will compare the current file to the file to load. If they are different, then the player will load the new file and perform the requested action. The player must already be opened before *fileToLoad* will work. Also refer to performance improvements for additional information.

For example, to set this attribute, connect the *selectedItem* attribute of a list box to *fileToLoad*.

isMuted (Boolean)

The *isMuted* attribute indicates whether the sound is on or off for the player.

For example, connect *isMuted* to the *selection* attribute of the mute button to ensure that the button's appearance matches the state of the player.

isPaused (Boolean)

The *isPaused* attribute indicates whether play is suspended for the player.

For example, connect *isPaused* to the *selection* attribute of the pause button to ensure that the button's appearance matches the state of the player.

lastError (MtLastError) - Read-only

The *lastError* attribute represents the last multimedia error that occurred. Typically, you do not make a connection to this attribute but instead, tear it off. By tearing it off, you gain access to information about the specific error that occurred, such as the error number and related message text.

For example, if you want to display a message box when an error occurs, tear off the *lastError* attribute. You can now connect the *errorOccured* event to the *display* action of the last error object to display the message box.

location (Integer) - Read-only

The *location* attribute is the time in MMTIME where the player is currently playing or positioned. See also the *totalTimeAsMMTime* attribute of *currentFile*.

For example, to have a slider control reflect the current position of the video file in OS/2, connect this attribute to the *value* attribute of the slider. Or, in OS/2 or Windows, connect *location* to the *object* attribute of a label.

self (MtVideoPlaybackWindow) - Read-only

The *self* attribute represents the digital video player itself.

For example, to tell the motion buttons what player to control, connect *self* to *player* of the motion buttons.

videoPlaybackWindow (MtVideoPlaybackWindow)

The *videoPlaybackWindow* attribute is where the video is to be displayed. If the value is *nil*, then the video is played in the default window. Otherwise, the video is displayed in the window represented by this object.

For example, connect *videoWindow* of a video playback window to this attribute to display the video in that window.

volume (Integer)

The *volume* attribute represents the current volume setting for the player.

For example, connect *volume* to the *value* of a slider.

Volume is supported in OS/2 but not in Windows.

Actions

Use the Digital Video Player actions when you want your application to tell the digital video player to do something.

close

The *close* action closes the player and frees any associated resources. After the player closes, it must be opened again before it can play a file. If the player is currently performing some work, that work will stop.

For example, connect to *close* when you no longer need the player, such as when your view is about to close.

fastForward

The *fastForward* action rapidly moves the position of the media forward. This connection is not needed if you made the *player-self* and *stateChanged-setEmphasis* connections.

For example, connect to *fastForward* when the fast forward button is *clicked*.

frameAdvance

The *frameAdvance* action moves the current position in the file forward by one frame. This connection is not needed if you made the *player-self* connection.

For example, connect to *frameAdvance* when the Frame Advance button is *clicked*.

frameReverse

The *frameReverse* action moves the current position in the file backward by one frame. This connection is not needed if you made the *player-self* connection.

For example, connect to *frameReverse* when the Frame Reverse button is *clicked*.

mute

The *mute* action silences the player. This connection is not needed if you made the *player-self* connection.

For example, connect to *mute* when the **Mute** toggle button is *clicked*.

mute is supported in OS/2 but not in Windows.

open

The *open* action opens and initializes the player. Each player initializes with different default files:

- For the digital video player, the digital video file is `new.avi`.
- For the audio wave player, the audio wave file is `new.wav`.

For example, connect to *open* when you are initializing your window, such as when the *aboutToOpenWidget* event occurs.

openFile

openFile: opens and initializes the player with the file described by the *aFileName* parameter.

For example, connect to this when you are initializing your window, such as when the *aboutToOpenWidget* event occurs.

pause

The *pause* action suspends playing of the player but preserves the current position in the file. This connection is not needed if you made the *player-self* connection.

For example, connect to *pause* when the **Pause** toggle button is *clicked*.

play

The *play* action starts playing the player. This connection is not needed if you made the *player-self* connection.

For example, connect to *play* when the play button is *clicked*.

playShortFile: fileName (String)

The *playShortFile:* action is useful for playing short digital video files. The action opens a player, plays the file described by the *fileName* parameter, and then closes the player. The action does not return until the entire file is played. Thus, for long files, it may appear that the machine is hung.

For example, connect the *ok* event of a file selection prompter to the *playShortFile:* action of a player.

rewind

The *rewind* action rapidly moves the position of the media backward. This connection is not needed if you made the *player-self* connection.

For example, connect to *rewind* when the Rewind button is *clicked*.

stop

The *stop* action halts the current action of the player.

For example, connect to *stop* when the stop button is *clicked*.

wrap

The *wrap* action toggles the wrap setting, which determines what the player does when the end or beginning of the media is reached. During play, when this attribute is set to *true*, the player continues play at the beginning of the media when the end is encountered. This connection is not needed if you made the *player-self* connection.

For example, connect to *wrap* when the **Wrap** toggle button is *clicked*.

Events

Use the digital video player events when you want your application to be notified when a change has occurred in the digital video player.

initialized

The *initialized* event occurs when the player finishes loading and querying the device for the attributes contained in the player.

For example, connect an action to this event to change from the wait pointer back to the arrow pointer.

majorErrorOccurred

The *majorErrorOccurred* event occurs when the player cannot fully initialize. One reason might be that the multimedia dynamic link library could not be found and loaded. Another reason might be that the player could not be found. You can connect an action to this event to inform the user that the player cannot initialize.

majorErrorOccurred is different from the *errorOccurred* event in the *lastError* object and does not generate error text. They are distinct so you can easily distinguish between a fatal error and a non-fatal error without having to write any code.

For example, connect to this event if you want to close your application or display a message to the user.

noVolumeRange

The *noVolumeRange* event occurs when the player does not support the full range of volume. Only 0 (off) and 100 (on) are supported.

For example, you can disable the volume slider with this event.

Volume is supported in OS/2 but not in Windows.

playWrapped

The *playWrapped* event occurs when the end of the media is encountered during a play operation and playing continued at the start of the media. Play will wrap only if *wrap* is set to *true*.

For example, you could connect to this event to update a slider which shows current position in a digital video or audio wave file.

reachedBegin

The *reachedBegin* event occurs when the beginning of the media is reached during play or other movement of the player.

For example, you can connect to this event to disable any buttons that are not available at the beginning of a digital video or audio wave file.

reachedEnd

The *reachedEnd* event occurs when the end of the media is reached during play or other movement of the player.

For example, you can connect to this event to disable any buttons that are not available at the end of a digital video or audio wave file.

stateChanged

The *stateChanged* event occurs when the state of the player changes. For example, the state goes from STOPPED to PLAYING when the play button is pressed.

For example, you could connect the *setEmphasis* action to this event.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

Opening and closing the application

When your application opens, you should make a connection to open the digital video player. The easiest way to open the digital video player is by connecting the *openedWidget* event of your window to the *openFile* action of the digital video player. Then, connect the name of the file you want to open to the *aFileName* parameter of the *openFile* connection.

When your application is finished working with the digital video player, we recommend that you close it. Then, when your application terminates, the player doesn't remain open. Closing the digital video player ensures that other applications that use digital video will have access to the player. One example is to connect the *aboutToCloseWidget* event of your window to the *close* action of the digital video player.

It is important that, once opened, a multimedia player close before it, or another player, opens again. Otherwise, your system might hang or another unexpected result might occur.

Error handling

The easiest way for you to add error handling to your application is to tear off the *lastError* attribute and connect the *errorOccured* event of *lastError* to the *display* action of *lastError*. Now, whenever an error occurs, a message box displays.

There are three different types of errors: critical, warning and information. You can control which errors are displayed in the message box by setting *errorThreshold*. There are three values that you can set:

1 Displays only critical errors.
2 Displays both critical and warning errors.
3 Displays critical, warning and information errors.

The default error threshold is 2. To change this, you can connect the *aboutToOpenWidget* event of your window to the *errorThreshold* attribute of *lastError*. Set the argument to be either 1, 2, or 3.

Performance improvement

Depending on how your application's user interface is designed, you may want to take advantage of the *fileToLoad* attribute. For example, you may build a user interface that contains both a list box which lists the digital video files and the motion buttons. If you want to play the selected file, you have two alternatives. The first is that every time you make a selection from the list, you open up the file. When the digital video player receives an open, it loads up the file and prepares for playback. For large files, this could be time consuming. A better alternative is to update the *fileToLoad* attribute as the user maneuvers through the list. The file will be loaded only when the user presses one of the buttons, such as Play.

What to watch for

Video compression types

Depending on the compression method used when the video file was recorded, it may or may not play. If the video file was recorded using the Intel Indeo 2.1 or above compression method, then the file should play on both OS/2 and Windows. If the video file was recorded using an IBM Ultimotion® compression method, then the file will not play on Windows unless you have the Ultimotion decompressor for Windows installed, which is available from IBM. If the video file was recorded using a Windows compression method, then the file will not play on OS/2.

Unnecessary connections

If you made the *player-self* connection, you do not need to make a *playPressed-play* connection.

Playing a short file

If you use the *playShortWaveFile* action, please note the following:

- The system volume control must be used to control the sound.
- The elapsed time does not update until the play ends.
- If the digital video player is already open, calling this action closes the player.

Chapter 9. Fast Forward Button

Select the fast forward button part to rapidly move the position of the media forward.

Part:

Category:
Multimedia

Class Name:
MtFastForwardButtonView

Attributes

Use the Fast Forward button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Fast Forward button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Fast Forward button events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

clicked

The *clicked* event occurs when the user presses and releases a button.

For example, depending on the button, connect *clicked* to the appropriate action in a player.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Wrapping

Fast forwarding through the media will not wrap to the beginning of the media if the wrap button is set. Fast forward will always stop when the end of the media is reached.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 10. Frame Advance Button

Select the frame advance button part to move the current position in the file forward by one frame.

Part:

Category:
Multimedia

Class Name:
MtFrameAdvanceButtonView

Attributes

Use the Frame Advance button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Frame Advance button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Frame Advance button events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

clicked

The *clicked* event occurs when the user presses and releases a button.

For example, depending on the button, connect *clicked* to the appropriate action in a player.

General advice

Recommended connections

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Wrapping

Stepping forward through the media will not wrap to the beginning of the media if the wrap button is set. Frame advance will always stop when the end of the media is reached.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 11. Frame Buttons

The Frame Buttons part is a group of buttons like those found on a video player. These buttons are used to control the frame movement of video.

The buttons that make up this button block are as follows:

- Frame Advance Button
- Frame Reverse Button

Part:

Category:
Multimedia

Class Name:
MtFrameButtons

Attributes

Use the Frame Buttons' attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Frame Buttons' actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Frame Buttons' events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

frameAdvancedPressed

The *frameAdvancePressed* event occurs when a user presses and releases the Frame Advance button. This is equivalent to the *clicked* event for the Frame Advance button.

For example, you can connect this event to navigate forward through database records.

frameReversePressed

The *frameReversePressed* event occurs when a user presses and releases the Frame Reverse button. This is equivalent to the *clicked* event for the Frame Reverse button.

For example, you can connect this event to navigate backward through database records.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Things you can specialize

Other uses for buttons

The buttons that are provided do not need to be used for multimedia actions. You can use them to perform other actions in your application. For example, you can connect the *frameAdvancePressed* event to the *getNextRow* action for a database object. This lets you have graphic buttons in your user interface instead of the standard text push buttons.

Chapter 12. Frame Reverse Button

Select the frame reverse button part to move the current position in the file backward by one frame.

Part:

Category:
Multimedia

Class Name:
MtFrameReverseButtonView

Attributes

Use the Frame Reverse button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Frame Reverse button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Frame Reverse button events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

clicked

The *clicked* event occurs when the user presses and releases a button.

For example, depending on the button, connect *clicked* to the appropriate action in a player.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Wrapping

Stepping backward through the media will not wrap to the end of the media if the wrap button is set. Frame reverse will always stop when the beginning of the media is reached.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 13. Last Error

The last error part represents the last error that took place in a multimedia part.

This part allows you to add error handling to your applications without having to write your own messages for errors that might occur through use of the multimedia parts. For example, the error text for a MMPM error is automatically filled in.

To access the last error's public interface, you must first add a part that has this attribute to your application such as the digital video player. Then, you tear off the *lastError* attribute from that part.

Part:

Category:
Multimedia

Class Name:
MtLastError

Attributes

This part allows you to add error handling to your applications without having to write your own messages for errors that might occur through use of the multimedia parts. For example, the error text for a MMPM error is automatically filled in.

classErrorIn (String) - Read-only

The *classErrorIn* attribute provides the name of the class name where the last error occurred. Knowing the class name tells you where to begin looking if an error condition occurs.

For example, connect the *classErrorIn* attribute to the *label* attribute of a static text field to see the class name where the error occurred.

errorText (String) - Read-only

The *errorText* attribute provides the text for the last error.

For example, connect this attribute to the *label* attribute of a static text field.

errorThreshold (Integer)

The *errorThreshold* attribute represents the level of errors that should be displayed in a message box. The *errorThreshold* can include:

1 Only critical errors
2 Both critical and warning errors
3 Critical, warning, and information errors

For example, to display all levels of errors, connect the *object* attribute of an edit field to this attribute. Then, when the application is running, type 3 into the edit field.

errorValue (Integer) - Read-only

The *errorValue* attribute is the value of the last error that occurred.

For example, connect this attribute to the *object* attribute of an edit field to display the error value that was returned.

methodErrorIn (String) - Read-only

The *methodErrorIn* attribute is the method name where the last error occurred. When used with *classErrorIn*, you know where the error took place and can start tracing back from there.

For example, connect the *methodErrorIn* attribute to the *label* attribute of a static text field.

Actions

Use the last error actions when you want your application to tell the last error object to do something. The following list contains the actions that are available in the last error object.

lastClearError

The *clearLastError* action resets the error variables to 0, which reflects a no error condition.

For example, you can connect to this action from any event in which you want to reset the last error information.

display

The *display* action displays a message box for the error that occurred.

For example, to display a message box, connect this to the *errorOccurred* event.

Events

Use the last error object events when you want your application to be notified when a change has occurred in the last error object. The following list contains the events that are available in the last error object.

errorOccurred

The *errorOccurred* event occurs when an error arises while a logical unit of work is being performed. Look at the attributes of the last error part for specific error information.

For example, you can connect this event to the *display* action to display a message box for the error.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Error handling

The easiest way for you to add error handling to your application is to tear off the *lastError* attribute and connect the *errorOccured* event of *lastError* to the *display* action of *lastError*. Now, whenever an error occurs, a message box displays.

There are three different types of errors: critical, warning and information. You can control which errors are displayed in the message box by setting *errorThreshold*. There are three values that you can set:

1 Displays only critical errors.
2 Displays both critical and warning errors.
3 Displays critical, warning and information errors.

The default error threshold is 2. To change this, you can connect the *aboutToOpenWidget* event of your window to the *errorThreshold* attribute of *lastError*. Set the argument to be either 1, 2, or 3.

Debugging

To help debugging, instead of bringing up a message box every time there is an error, place the attributes in your window. Then, every time there is an error, these fields are updated and you do not have a message box with which to interact.

Chapter 14. Media Control Interface

The Media Control Interface (MCI) part provides a programming interface to the operating system's multimedia programming interface. This part is not on the palette but you can add it by selecting **Add Part** from the **Options** menu.

Unlike the other parts where much, if not all, of the code has been written for the function you need, the MCI part allows you to write your own code and send it to the MCI. You must be familiar with the operating system's MCI to make use of this part.

To use this part, simply connect to the *sendString:* action of the MCI part and pass the MCI string that you want to be run. Or, write a simple script which makes use of the *sendString:* method. The *returnString* attribute contains the return value from the *mciSendString* function call.

Part:
MCI

Class Name:
MtMCI

Attributes

Use the Media Control Interface attributes when you want your application to reflect some information that is maintained in the Media Control Interface. The following list contains the attributes that are available.

lastError (MtLastError) - Read-only

The *lastError* attribute represents the last multimedia error that occurred. Typically, you do not make a connection to this attribute but instead, tear it off. By tearing it off, you gain access to information about the specific error that occurred, such as the error number and related message text.

For example, if you want to display a message box when an error occurs, tear off the *lastError* attribute. You can now connect the *errorOccured* event to the *display* action of the last error object to display the message box.

returnString (String) - Read-only

The *returnString* attribute is the return string from the MCI command.

For example, connect this attribute to the *labelString* attribute of a label part.

Actions

Use the Media Control Interface actions when you want your application to tell the Media Control Interface to do something.

send: aCommandString (String)

The *send:* action sends *aCommandString* to the MCI. The return string can be found in the *returnString* attribute and any information can be found in the *lastError*

attribute. If you specified the *notify* flag in your string, then make an event-to-script connection, with *notified:with:* as the event, to access the notification information.

Events

Use the Media Control Interface events when you want your application to be notified when a change has occurred in the Media Control Interface.

notified: mp1 (MPARAM) with: mp2 (MPARAM)

The *notified:with:* event occurs when the *notify* flag is specified in your MCI command string. *mp1* and *mp2* contain the MCI information as specified in your MCI documentation. The actual values vary between OS/2 and Windows.

If you connect this event to a script, the first two parameters of that script will be *mp1* and *mp2*. The same holds true if making an event-to-action connection.

Chapter 15. Motion Buttons

The Motion Buttons part is a group of buttons like those found on your video cassette recorder or compact disc player at home. These buttons are used to control the standard motion of the media like play, fast forward, and rewind.

The buttons that make up this button block are as follows:

- Play Button
- Stop Button
- Rewind Button
- Fast Forward Button
- Pause Button

Part: 

Class Name:
MtMotionButtons

Attributes

Use the Motion Buttons' attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

isPaused (Boolean)

The *isPaused* attribute indicates whether play is suspended for the player.

For example, connect *isPaused* to the *selection* attribute of the pause button to ensure that the button's appearance matches the state of the player.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Motion Buttons' actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Motion Buttons' events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

fastForwardPressed

The *fastForwardPressed* event occurs when the user presses and releases the Fast Forward button. This is equivalent to the *clicked* event for the Fast Forward button.

pausePressed

The *pausePressed* event occurs when the user presses and releases the Pause button. This is equivalent to the *clicked* event for the Pause button.

playPressed

The *playPressed* event occurs when the user presses and releases the Play button. This is equivalent to the *clicked* event for the Play button.

rewindPressed

The *rewindPressed* event occurs when the user presses and releases the Rewind button. This is equivalent to the *clicked* event for the Rewind button.

stopPressed

The *stopPressed* event occurs when the user presses and releases the Stop button. This is equivalent to the *clicked* event for the Stop button.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Things you can specialize

Other uses for buttons

The buttons that are provided don't necessarily have to be used for multimedia actions. You can use them to perform other actions in your application. For example, you can connect the *playPressed* event to the *getNextRow* action for a database object. This lets you have graphic buttons in your user interface instead of the standard text push buttons.

Chapter 16. Mute Button

Select the mute button to toggle the sound on or off for the player.

Part:

Class Name:

MtMuteButtonView

Attributes

Use the Mute button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

selection

The *selection* attribute keeps the button matched with the state of the player. This is similiar to the behavior of the *clicked* event for push buttons. This connection is not necessary if you make the *setEmphasis* connection.

For example, make this connection to keep track of the mute state of the player.

Actions

Use the Mute button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 17. Pause Button

Select the pause button to suspend and resume the playing and recording of a player.

Part:

Class Name:

MtPauseButtonView

Attributes

Use the Pause button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

selection

The *selection* attribute keeps the button matched with the state of the player. This is similiar to the behavior of the *clicked* event for push buttons. This connection is not necessary if you make the *setEmphasis* connection.

For example, make this connection to keep track of the mute state of the player.

Actions

Use the Pause button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 18. Play Button

Select the play button to play the media.

Part:

Class Name:

MtPlayButtonView

Attributes

Use the Play button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Play button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Play button events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

clicked

The *clicked* event occurs when the user presses and releases a button.

For example, depending on the button, connect *clicked* to the appropriate action in a player.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 19. Record Button

Select the record button if you want to record audio into a file. This starts recording from the input device, starting at the current location and overwriting any existing data.

Part:

Class Name:

MtRecordButtonView

Attributes

Use the Record button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Record button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Record button events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

clicked

The *clicked* event occurs when the user presses and releases a button.

For example, depending on the button, connect *clicked* to the appropriate action in a player.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 20. Rewind Button

Select the rewind button to rapidly move the position of the media backward.

Part:

Class Name:

MtRewindButtonView

Attributes

Use the Rewind button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Rewind button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Rewind button events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

clicked

The *clicked* event occurs when the user presses and releases a button.

For example, depending on the button, connect *clicked* to the appropriate action in a player.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Wrapping

Rewinding through the media will not wrap to the end of the media if the wrap button is set. Rewind will always stop when the beginning of the media is reached.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 21. Stop Button

Select the stop button to halt the current action of the player.

Part:

Class Name:

MtStopButtonView

Attributes

Use the Stop button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

Actions

Use the Stop button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

Events

Use the Stop button events when you want your application to be notified when the button has done something significant. The following list contains the events that are available.

clicked

The *clicked* event occurs when the user presses and releases a button.

For example, depending on the button, connect *clicked* to the appropriate action in a player.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Chapter 22. Timer

The timer part signals an event after a certain amount of time has elapsed.

This is useful when you want to introduce the concept of time into your application. For example, you might want to toggle through a series of images, each one being displayed for a certain amount of time before the next one appears.

Part:

Class Name:
MtTimer

Attributes

Use the timer attributes when you want your application to reflect some information that is maintained in the timer. The following list contains the attributes that are available.

length (Integer)

The *length* attribute is the length, in milliseconds, before the timer fires.

report (Boolean)

The *report* attribute sets whether the timer should restart after it fires.

userData (Object)

The *userData* attribute is data that you want passed when the *timerFired* event occurs. This data can be anything and will retain its value across multiple *timerFired* events.

Actions

Use the timer actions when you want your application to tell the timer to do something.

reset

The *reset* action restarts the timer by stopping and then restarting it.

start

The *start* action starts the timer. The timer fires after the amount of time specified in *length* expires.

For example, connect the *openedWidget* event to this action.

stop

The *stop* action halts the timer from firing.

For example, connect the *aboutToCloseWidget* event to the *stop* action.

Events

Use the timer events when you want your application to be notified when a change has occurred in the timer.

timerFired: userData (Object)

The *timerFired:* event occurs when the *length* of the timer expires and the timer fires.

Chapter 23. Video Playback Window

Select the video playback window part to display video data in a window other than the default video window. This allows you to embed video with other controls in one window. The default video window is created by the operating system and is a desktop window.

Part: 

Class Name:
MtVideoPlaybackWindow

Attributes

Use the video playback window attributes when you want your application to reflect some information that is maintained in the video playback window. The following list contains the attributes that are available in the video playback window.

recomputeSize (Boolean)

The *recomputeSize* attribute indicates whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

videoWindow (MtVideoPlaybackWindow)

The *videoWindow* attributes is the video playback window object itself.

For example, to display the video in this window, connect it to the *videoPlaybackWindow* attribute of a digital video player.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Displaying video in the video playback window

The only necessary connection connects the *videoWindow* attribute of the video playback window to the *videoPlaybackWindow* attribute of a digital video player.

What to watch for

Resizing the video playback window

When video is played, it can play in three different sizes: normal, half, and double. Regardless of the size of the window, the video will not resize itself to the dimensions of the window. In all likelihood, the video will be clipped and you will not be able to see the entire video. Therefore, when the *recomputeSize* attribute is *true*, the following algorithm is used to change the video playback window size:

1. If the video window is larger than double the video size, then resize the window to be the double size of the video.

2. If the video window is larger than the normal video size, then resize the window to be the normal size of the video.
3. If the video window is larger than half the video size, then resize the window to be the half size of the video.
4. If the above conditions fail, then do not resize the window but trigger an error message which can be retrieved from the last error object.

Chapter 24. Wrap Button

Select the wrap button to turn wrap on or off.

Toggling the wrap setting determines what the player does when the end or beginning of the media is reached. During play, when this attribute is set to *true*, the player continues play at the beginning of the media when the end of the media is encountered.

Part:

Class Name:

MtWrapButtonView

Attributes

Use the Wrap button attributes when you want your application to reflect some information that is maintained in the button. The following list contains the attributes that are available.

player (MtMediaDevice)

The *player* attribute is the media device that the button will control. You can control a media device by connecting this attribute to the *self* attribute of the player.

For example, if you connect a button's *player* attribute to the *self* attribute of a digital audio player then, when the button is clicked, the appropriate action is performed.

selection

The *selection* attribute keeps the button matched with the state of the player. This is similiar to the behavior of the *clicked* event for push buttons. This connection is not necessary if you make the *setEmphasis* connection.

For example, make this connection to keep track of the mute state of the player.

Actions

Use the Wrap button actions when you want the button to do something, such as disabling or enabling itself. The following list contains the actions that are available.

disable

The *disable* action makes the button unavailable for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

enable

The *enable* action makes the button available for selection. This connection is not necessary if you make the *setEmphasis* connection. Also see the recommended connections section for additional information.

setEmphasis

The *setEmphasis* action enables or disables one or more buttons based on the state of the player or scrollable part.

For example, to disable the appropriate motion buttons, connect the *stateChanged* event in the player to *setEmphasis*.

General advice

Recommended connections

You can make any connection between objects. The following explains certain key connections that can be made.

Getting the buttons to work

Only two connections are necessary. First, connect the *player* attribute of any button or button block to the *self* attribute of the audio wave player or digital video player. When this connection exists, the button can communicate with the player and take action when a button is pressed.

Next, connect the *stateChanged* event of the player to the *setEmphasis* action of the buttons. This enables and disables the buttons based on the state of the player. This connection is optional but gives your application a common look and feel.

What to watch for

Unnecessary connections

If you made the *player-self* connection, you do not need to make a specific connection to the button.

Fast forward and rewind

Fast forward and rewind always stop at the beginning or end of the media. They do not wrap, regardless of the wrap setting.

Things you can specialize

Changing the appearance of the buttons

If you would like to use a different bitmap on the face of any button, you can open its settings page in the Composition Editor and change its *graphicsDescriptor* setting. You need to create a dynamic link library that contains the bitmap.

Part 3. Multimedia Help

Chapter 25. Multimedia category

The Multimedia category contains parts for audio and video playback that you can add to your applications.

The parts in the Multimedia category are as follows:

Audio Wave Player

Digital Video Player

Video Playback Window

Timer

Motion Buttons

Frame Buttons

Record Button

Wrap Button

Mute Button

Play Button

Stop Button

Rewind Button

Fast Forward Button

Pause Button

Frame Reverse Button

Frame Advance Button

Also, the **Media Control Interface** part (

) is available through the **Add part** dialog.

Audio Wave Player

Use the Audio Wave Player part to play or record digital audio using files.

While **audio** refers to the sound waves that have a perceived effect on the human ear, **wave form** refers to the digital representation of the original audio sound wave.

Graphic

Class Name
MtAudioWavePlayer

Digital Video Player

Use the Digital Video Player part to play digital video using files. The files can contain both video and audio data or just video data. The video can be played either in a separate window on the desktop or imbedded in an existing window like a dialog.

Digital video can come in different formats. Look at your operating system multimedia documentation to see what multimedia input/output (MMIO) procedures are supported. For example, OS/2 Warp supports:

Audio/Video Interleaved (AVI)
Software motion video that does not require specific hardware to play the video.

MPEG
Video that requires hardware to be played. For example, you can use a ReelMagic adapter to play MPEG files.

FLI and FLI
Animation files.

Graphic

Class Name
MtDigitalVideoPlayer

Video Playback Window

Use the Video Playback Window part to display video data in a window other than the default video window. This allows you to embed video with other controls in one window. The default video window is created by the operating system and is a desktop window.

Graphic

Class Name
MtVideoPlaybackWindow

Video Playback Window Properties

The Video Playback Window part has the following properties:

- backgroundColor
- backgroundGraphicsDescriptor
- borderWidth
- buttonFontName
- enabled
- foregroundColor
- fractionBase
- framingSpec
- helpFile

- helpKeysId
- helpTitle
- helpTopicId
- horizontalSpacing
- hoverHelpEnabled
- labelFontName
- marginHeight
- marginWidth
- partName
- recomputeSize
- rubberPositioning
- tabGroup
- textFontName
- traversalOn
- verticalSpacing
- wallpaperStyle

backgroundColor
Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor
Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth
Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

buttonFontName
Use the *buttonFontName* property to specify the font used for the buttons in the part. If a font is not specified, the font specified on the *textFontName* property is used for the buttons in the part.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

fractionBase
Use the *fractionBase* property to specify the denominator used in the fraction to calculate the percentage for proportional spacing. The default value is 100.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

horizontalSpacing
Use the *horizontalSpacing* property to specify the default horizontal space between the part and its children. The default value is *0*.

hoverHelpEnabled
Use the *hoverHelpEnabled* property to indicate whether you want hover help to be provided for this part and all of its children. The default is *false*.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

marginHeight
Use the *marginHeight* property to specify the amount of vertical space between the part and its children.

marginWidth
Use the *marginWidth* property to specify the amount of horizontal space between the part and its children that are not attached.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

rubberPositioning
Use the *rubberPositioning* property to specify whether child parts have their positions converted to proportional attachments. If *rubberPositioning* is set to *true*, all children parts with no attachments on their top, bottom, left, or right, have their initial top and left positions converted to proportional attachments. If *rubberPositioning* is set to *false*, then the initial top and left positions are attached to the parent.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

textFontName
Use the *textFontName* property to specify the font used for child text parts of a part. If you do not specify a font using the *buttonFontName* or *labelFontName* properties, the font specified for the *textFontName* property is used.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

verticalSpacing
Use the *verticalSpacing* property to specify the default vertical space between the part and its children.

wallpaperStyle
Use the *wallpaperStyle* property to specify how you want the wallpaper images displayed. Possible values are—

- *AbtNormalWallpaper*
- *AbtScaledWallpaper*
- *AbtTiledWallpaper*

Timer

The Timer part signals an event after a certain amount of time has elapsed.

You use a timer to introduce the concept of time into your application. For example, you might want to toggle through a series of images, each one being displayed for a certain amount of time before the next one appears.

Graphic

Class Name

MtTimer

Timer Properties

The Timer part has the following properties:

length Use the *length* property to specify the number of milliseconds before the timer fires.

partName

Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

repeat Use the *repeat* property to specify whether the timer should restart after it fires.

userData

Use the *userData* property to specify the data passed when the *timerFired* event occurs. The data can be anything and will retain its value across multiple *timerFired* events.

Timer Settings

Use the MtTimer Settings window to provide the information necessary to use the timer.

Timer fires after

In the **Timer fires after** field, type the number of milliseconds before the timer is to trigger the *timerFired:* event.

Repeat

Select **Repeat** if you want the timer to continuously trigger the *timerFired:* event. The event will fire every X milliseconds, based on what was specified in the **Timer fires after** entry field.

User data

In the **User data** field, type any data that is to be passed to you when the *timerFired:* event is triggered.

Motion Buttons

The Motion Buttons part is a group of buttons like those found on your video cassette recorder or compact disc player at home. These buttons are used to control the standard motion of the media like play, fast forward, and rewind.

The buttons that make up this button block are as follows:

Play Button

Stop Button

Rewind Button

Fast Forward Button

Pause Button

Graphic

Class Name

MtMotionButtons

Motion Buttons Properties

The Motion Buttons part has the following properties:

- backgroundColor
- backgroundGraphicsDescriptor
- borderWidth
- buttonFontName
- enabled
- foregroundColor
- fractionBase
- framingSpec
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- horizontalSpacing
- hoverHelpEnabled
- isPaused
- labelFontName
- marginHeight
- marginWidth
- partName
- rubberPositioning
- tabGroup
- textFontName

- traversalOn
- verticalSpacing
- wallpaperStyle

backgroundColor
Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor
Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth
Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

buttonFontName
Use the *buttonFontName* property to specify the font used for the buttons in the part. If a font is not specified, the font specified on the *textFontName* property is used for the buttons in the part.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

fractionBase
Use the *fractionBase* property to specify the denominator used in the fraction to calculate the percentage for proportional spacing. The default value is 100.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

horizontalSpacing
Use the *horizontalSpacing* property to specify the default horizontal space between the part and its children. The default value is *0*.

hoverHelpEnabled
Use the *hoverHelpEnabled* property to indicate whether you want hover help to be provided for this part and all of its children. The default is *false*.

isPaused
Use the *isPaused* property to indicate whether the application is pausing or whether the toggle indicates the application should pause. *true* is specified by default.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

marginHeight
Use the *marginHeight* property to specify the amount of vertical space between the part and its children.

marginWidth
Use the *marginWidth* property to specify the amount of horizontal space between the part and its children that are not attached.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

rubberPositioning
Use the *rubberPositioning* property to specify whether child parts have their positions converted to proportional attachments. If *rubberPositioning* is set to *true*, all children parts with no attachments on their top, bottom, left, or right, have their initial top and left positions converted to proportional attachments. If *rubberPositioning* is set to *false*, then the initial top and left positions are attached to the parent.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

textFontName
Use the *textFontName* property to specify the font used for child text parts of a part. If you do not specify a font using the *buttonFontName* or *labelFontName* properties, the font specified for the *textFontName* property is used.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

verticalSpacing
Use the *verticalSpacing* property to specify the default vertical space between the part and its children.

wallpaperStyle
Use the *wallpaperStyle* property to specify how you want the wallpaper images displayed. Possible values are—
- *AbtNormalWallpaper*
- *AbtScaledWallpaper*
- *AbtTiledWallpaper*

Frame Buttons

The Frame Buttons part is a group of buttons like those found on a video disc player. You use these buttons to control the frame movement of video.

The buttons that make up this button block are as follows:

Frame Advance Button

Frame Reverse Button

Graphic

Class Name

MtFrameButtons

Frame Buttons Properties

The Frame Buttons part has the following properties:

- backgroundColor
- backgroundGraphicsDescriptor
- borderWidth
- buttonFontName
- enabled
- foregroundColor
- fractionBase
- framingSpec
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- horizontalSpacing
- hoverHelpEnabled
- labelFontName
- marginHeight
- marginWidth
- partName
- rubberPositioning
- tabGroup
- textFontName
- traversalOn
- verticalSpacing
- wallpaperStyle

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor

Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

buttonFontName
Use the *buttonFontName* property to specify the font used for the buttons in the part. If a font is not specified, the font specified on the *textFontName* property is used for the buttons in the part.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

fractionBase
Use the *fractionBase* property to specify the denominator used in the fraction to calculate the percentage for proportional spacing. The default value is 100.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

horizontalSpacing
Use the *horizontalSpacing* property to specify the default horizontal space between the part and its children. The default value is *0*.

hoverHelpEnabled
Use the *hoverHelpEnabled* property to indicate whether you want hover help to be provided for this part and all of its children. The default is *false*.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

marginHeight
Use the *marginHeight* property to specify the amount of vertical space between the part and its children.

marginWidth
Use the *marginWidth* property to specify the amount of horizontal space between the part and its children that are not attached.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

rubberPositioning
Use the *rubberPositioning* property to specify whether child parts have their

positions converted to proportional attachments. If *rubberPositioning* is set to *true*, all children parts with no attachments on their top, bottom, left, or right, have their initial top and left positions converted to proportional attachments. If *rubberPositioning* is set to *false*, then the initial top and left positions are attached to the parent.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

textFontName
Use the *textFontName* property to specify the font used for child text parts of a part. If you do not specify a font using the *buttonFontName* or *labelFontName* properties, the font specified for the *textFontName* property is used.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

verticalSpacing
Use the *verticalSpacing* property to specify the default vertical space between the part and its children.

wallpaperStyle
Use the *wallpaperStyle* property to specify how you want the wallpaper images displayed. Possible values are—
- *AbtNormalWallpaper*
- *AbtScaledWallpaper*
- *AbtTiledWallpaper*

Record Button

Use the Record Button part to record audio into a file. Selecting the button records from the input device, starting at the current location and overwriting any existing data.

Graphic

Class Name
MtRecordButtonView

Record Button Properties

The Record Button part has the following properties:
- alignment
- backgroundColor
- borderWidth
- converter
- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId

- helpTitle
- helpTopicId
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- pushButton3DLookEnabled
- recomputeSize
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT

Left alignment

XmRIGHT

Right alignment

XmCENTER

Center alignment

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor

Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter

Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor

Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec

Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled

Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName
Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.
XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

pushButton3DLookEnabled
Use the *pushButton3DLookEnabled* property to specify whether the button is shaded to look three-dimensional. The property is *true* by default, enabling the 3D look.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Wrap Button

Use the Wrap Button part to turn wrap on or off. Toggling the wrap setting determines what the player does when the end or beginning of the media is reached. During play, when the wrap setting is set to *true*, the player continues play at the beginning of the media when the end of the media is encountered.

Graphic

Class Name
MtWrapButtonView

Wrap Button Properties

The Wrap Button part has the following properties:

- alignment
- backgroundColor
- borderWidth
- converter

- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- indicatorOn
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- recomputeSize
- selection
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT

Left alignment

XmRIGHT

Right alignment

XmCENTER

Center alignment

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter

Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor

Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec
Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName
Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

indicatorOn
Use the *indicatorOn* property to specify that the check box is drawn to the left of the check box text, icon, or bitmap; otherwise space is not allocated for the indicator and it is not displayed.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.

XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

selection
Use the *selection* property to keep the button matched with the state of the player. This is similiar to the behavior of the *clicked* event for push buttons. This connection is not necessary if you make the *setEmphasis* connection.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Mute Button

Use the Mute Button part to toggle the sound on or off for the player.

Graphic

Class Name

MtMuteButtonView

Mute Button Properties

The Mute Button part has the following properties:

- alignment
- backgroundColor
- borderWidth
- converter
- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- indicatorOn
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- recomputeSize
- selection
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT

Left alignment

XmRIGHT

Right alignment

XmCENTER

Center alignment

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter
Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor
Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec
Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName
Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

indicatorOn
Use the *indicatorOn* property to specify that the check box is drawn to the left of the check box text, icon, or bitmap; otherwise space is not allocated for the indicator and it is not displayed.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.

XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

selection
Use the *selection* property to keep the button matched with the state of the player. This is similiar to the behavior of the *clicked* event for push buttons. This connection is not necessary if you make the *setEmphasis* connection.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Play Button

Use the Play Button part to play the media.
Graphic

Class Name
MtPlayButtonView

Play Button Properties

The Play Button part has the following properties:
- alignment
- backgroundColor
- borderWidth
- converter
- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- pushButton3DLookEnabled
- recomputeSize
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT
Left alignment

XmRIGHT
Right alignment

XmCENTER
Center alignment

backgroundColor
Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor
Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth
Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter
Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor
Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec
Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName
Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.

XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

pushButton3DLookEnabled
Use the *pushButton3DLookEnabled* property to specify whether the button is shaded to look three-dimensional. The property is *true* by default, enabling the 3D look.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Stop Button

Use the Stop Button to halt the current action of the player.

Graphic

Class Name
MtStopButtonView

Stop Button Properties

The Stop Button part has the following properties:

- alignment
- backgroundColor
- borderWidth
- converter
- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- pushButton3DLookEnabled
- recomputeSize
- tabGroup
- traversalOn

alignment
Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT
Left alignment
XmRIGHT
Right alignment
XmCENTER
Center alignment

backgroundColor
Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor
Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth
Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter
Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor
Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec
Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName
Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.

XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

pushButton3DLookEnabled
Use the *pushButton3DLookEnabled* property to specify whether the button is shaded to look three-dimensional. The property is *true* by default, enabling the 3D look.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Rewind Button

Use the Rewind Button to rapidly move the position of the media backward.

Graphic

Class Name
MtRewindButtonView

Rewind Button Properties

The Rewind Button part has the following properties:

- alignment
- backgroundColor
- borderWidth
- converter
- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- pushButton3DLookEnabled

- recomputeSize
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT

Left alignment

XmRIGHT

Right alignment

XmCENTER

Center alignment

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor

Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter

Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor

Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec

Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled

Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName

Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor

Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec

Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor

Use the *graphicsDescriptor* property to specify an icon, bitmap, or image

that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.

XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

pushButton3DLookEnabled
Use the *pushButton3DLookEnabled* property to specify whether the button is shaded to look three-dimensional. The property is *true* by default, enabling the 3D look.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Fast Forward Button

Use the Fast Forward Button to rapidly move the position of the media forward.
Graphic

Class Name
MtFastForwardButtonView

Fast Forward Button Properties

The Fast Forward Button part has the following properties:

- alignment
- backgroundColor
- borderWidth
- converter
- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight

- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- pushButton3DLookEnabled
- recomputeSize
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT

Left alignment

XmRIGHT

Right alignment

XmCENTER

Center alignment

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor

Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter

Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor

Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec

Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled

Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName

Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor

Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.

XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to

select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName

Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

pushButton3DLookEnabled

Use the *pushButton3DLookEnabled* property to specify whether the button is shaded to look three-dimensional. The property is *true* by default, enabling the 3D look.

recomputeSize

Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

tabGroup

Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn

Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Pause Button

Use the Pause Button to suspend and resume the playing and recording of a player.

Graphic

Class Name

MtPauseButtonView

Pause Button Properties

The Pause Button part has the following properties:

- alignment
- backgroundColor
- borderWidth
- converter
- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile

- helpKeysId
- helpTitle
- helpTopicId
- indicatorOn
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- recomputeSize
- selection
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT

Left alignment

XmRIGHT

Right alignment

XmCENTER

Center alignment

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter

Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor

Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec

Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled

Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName

Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

indicatorOn
Use the *indicatorOn* property to specify that the check box is drawn to the left of the check box text, icon, or bitmap; otherwise space is not allocated for the indicator and it is not displayed.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.
XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.
XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

selection
Use the *selection* property to keep the button matched with the state of the player. This is similiar to the behavior of the *clicked* event for push buttons. This connection is not necessary if you make the *setEmphasis* connection.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Frame Reverse Button

Use the Frame Reverse button to move the current position in the file backward by one frame.

Graphic

Class Name
MtFrameReverseButtonView

Frame Reverse Button Properties

The Frame Reverse Button part has the following properties:

- alignment
- backgroundColor
- borderWidth
- converter

- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- pushButton3DLookEnabled
- recomputeSize
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT

Left alignment

XmRIGHT

Right alignment

XmCENTER

Center alignment

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor

Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels, that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter

Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor

Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or

image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec
Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName
Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.

XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

pushButton3DLookEnabled
Use the *pushButton3DLookEnabled* property to specify whether the button is shaded to look three-dimensional. The property is *true* by default, enabling the 3D look.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Frame Advance Button

Use the Frame Advance Button to move the current position in the file forward by one frame.

Graphic

Class Name

MtFrameAdvanceButtonView

Frame Advance Button Properties

The Frame Advance Button part has the following properties:

- alignment
- backgroundColor
- borderWidth
- converter
- disabledGraphicsDescriptor
- dragDropSpec
- enabled
- fontName
- foregroundColor
- framingSpec
- graphicsDescriptor
- helpFile
- helpKeysId
- helpTitle
- helpTopicId
- labelMarginBottom
- labelMarginHeight
- labelMarginLeft
- labelMarginRight
- labelMarginTop
- labelMarginWidth
- labelType
- mnemonic
- object
- partName
- pushButton3DLookEnabled
- recomputeSize
- tabGroup
- traversalOn

alignment

Use the *alignment* property to specify that the text, icon, or bitmap can be aligned either left, right, or centered. On some platforms, the alignment for push buttons, toggle buttons, and cascade buttons cannot be set and is ignored. The possible values are as follows:

XmLEFT

Left alignment

XmRIGHT

Right alignment

XmCENTER

Center alignment

backgroundColor

Use the *backgroundColor* property to name the RGB color for the background of the part.

backgroundGraphicsDescriptor

Use the *backgroundGraphicsDescriptor* property to specify the icon, bitmap, or image that is used for the background of the part. If the part has a *labelType* property, the *labelType* property must be an icon, bitmap, or image or the *backgroundGraphicsDescriptor* property is ignored.

borderWidth

Use the *borderWidth* property to specify the width of the border, in pixels,

that surrounds the part on all four sides. A width of zero means that a border does not show. On some platforms, the border width is limited to 1 pixel and any nonzero value is set to 1.

converter
Use the *converter* property to specify the converter to use for the part. A converter is an object that manages the conversion of other objects to and from their display format for certain visual parts, such as a Text part.

disabledGraphicsDescriptor
Use the *disabledGraphicsDescriptor* property to specify an icon, bitmap, or image used to indicate that the part is disabled. The *labelType* property must be an icon, bitmap, or image or the *disabledGraphicsDescriptor* property is ignored.

dragDropSpec
Use the *dragDropSpec* property to specify drag and drop information for the part. You can tear-off the *dragDropSpec* property to access its properties and events.

enabled
Use the *enabled* property to specify whether a part is available for selection. If *true*, the part's contents are available for selection.

fontName
Use the *fontName* property to name the font to use for the text displayed in the part.

foregroundColor
Use the *foregroundColor* property to name the RGB color for the foreground of the part.

framingSpec
Use the *framingSpec* property to represent values specified for the edge specifications of the part. The edge specifications determine the width, height, and position of the part.

graphicsDescriptor
Use the *graphicsDescriptor* property to specify an icon, bitmap, or image that is used instead of text for the label. The *labelType* property must be XmPIXMAP or the *graphicsDescriptor* property is ignored.

helpFile
Use the *helpFile* property to name the help file associated with the part.

helpKeysId
Use the *helpKeysId* property to specify the ID of the keys help associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

helpTitle
Use the *helpTitle* property to specify the title of the help window associated with the part.

helpTopicId
Use the *helpTopicId* property to specify the ID of the help topic associated with the part. The ID can be a string or an integer value. The ID must exist in the help file associated with the part.

labelFontName
Use the *labelFontName* property to specify the font used for the labels of the part. If a font is not specified, the font specified on the *textFontName* property is used.

labelMarginBottom
Use the *labelMarginBottom* property to specify the amount of space between the label and the bottom margin.

labelMarginHeight
Use the *labelMarginHeight* property to specify the amount of space between the label and the bottom edge of the top shadow and the top edge of the bottom shadow.

labelMarginLeft
Use the *labelMarginLeft* property to specify the amount of space between the label and the left margin.

labelMarginRight
Use the *labelMarginRight* property to specify the amount of space between the right margin and the label of the part.

labelMarginTop
Use the *labelMarginTop* property to specify the amount of space between the the label and the top margin.

labelMarginWidth
Use the *labelMarginWidth* property to specify the amount of blank space between the label and the right edge of the left shadow and the left edge of the right shadow.

labelType
Use the *labelType* property to specify the type of label you want. The possible values are as follows:

XmICON
The icon defined by *graphicsDescriptor* is displayed.

XmPIXMAP
The graphic defined by *graphicsDescriptor* is displayed.

XmSTRING
The string defined by *labelString* is displayed.

mnemonic
Use the *mnemonic* property to specify a character that the user can type to select an item. A mnemonic character is displayed with an underscore (_) and is usually used for buttons belonging to menus.

object The *object* property is the data that the part represents. If the part represents non-string data, use this property; otherwise use the *labelString* or *string* property, if either is available. For example, you can use the *object* property with a converter to display the label of a button.

partName
Use the *partName* property to name the part. Every part dropped on the free-form surface is assigned a name that is unique from all other parts on the layout surface. For example, a push button is assigned a name such as "Push Button1". To change the name of a part, specify a new name for the *partName* property. Or, select **Change Name** from the pop-up menu of the part and enter a new name in the displayed dialog.

pushButton3DLookEnabled
Use the *pushButton3DLookEnabled* property to specify whether the button is shaded to look three-dimensional. The property is *true* by default, enabling the 3D look.

recomputeSize
Use the *recomputeSize* property to indicate whether the video playback window should automatically resize itself to fit the closest size of the video file that is loaded. The default value is *true*.

tabGroup
Use the *tabGroup* property to specify whether a part is included as a tab stop so users can use the **Tab** and **Backtab** keys to access the part.

traversalOn
Use the *traversalOn* property to specify whether users can access a part using the cursor movement keys. Specifying *true* indicates that the cursor movements keys can be used to access the part.

Media Control Interface

The Media Control Interface (MCI) part provides a programming interface to the operating system's multimedia programming interface. This part is not on the palette but you can add it by selecting **Add Part** from the **Options** menu.

Unlike the other parts where much, if not all, of the code has been written for the function you need, the MCI part allows you to write your own code and send it to the MCI. You must be familiar with the operating system's MCI to make use of this part.

To use this part, simply connect to the *sendString:* action of the MCI part and pass the MCI string that you want to be run. Or, write a simple script which makes use of the *sendString:* method. The *returnString* attribute contains the return value from the *mciSendString* function call.

Graphic

Class Name

MtMCI

To display the pop-up menu for the part, click with mouse button 2 on the part.

Audio Waveform File

The audio waveform file represents the actual medium that is being played by the audio wave player.

To access the audio waveform file's public interface, you must first add an audio wave player part to your application. Then, you tear off the *currentFile* attribute from the player.

Graphic

Class Name

MtAudioWave

Digital Video File

The digial video file represents the actual medium that is being played by the digital video player.

To access the digital video file's public interface, you must first add a Digital Video Player part to your application. Then, you tear off the *currentFile* attribute from the player.

Graphic

Class Name

MtDigitalVideo

Last Error

The last error part represents the last error that took place in a multimedia part.

This part allows you to add error handling to your applications without having to write your own messages for errors that might occur through use of the multimedia parts. For example, the error text for a MMPM error is automatically filled in.

To access the last error's public interface, you must first add a part that has this attribute to your application such as the Digital Video Player part. Then, you tear off the *lastError* attribute from that part.

Graphic

Class Name

MtLastError

Chapter 26. Pop-up menu for the Multimedia Parts

The pop-up menu for the multimedia parts contains the following choices:

- **Open Settings**
- **Edit Part**
- **Promote Part Feature**
- **View Parts List**
- **Change Name**
- **Delete**
- **Layout**
- **Become Primary Part**
- **Connect**
- **Browse Connections**
- **Reorder Connections From**
- **Quick Form**
- **Tear-Off Attribute**
- **Create Deferred Update Part**

Open Settings

Select **Open Settings** to display the settings for the selected parts. The Settings view displays the attributes and their initial values for the part. If multiple parts are selected when you select **Open Settings**, a settings notebook for each selected part opens in a different window. You can use the Settings view to change the values of the attributes that set up the initial appearance or behavior you need for the part.

Edit Part

Select **Edit Part** to open the editing view that is appropriate for the selected parts or the part whose pop-up menu you displayed. If the part is a visual part built with VisualAge, the Composition Editor for that part opens. Otherwise, the Script Editor for that part opens.

Promote Part Feature

Select **Promote Part Feature** to make the private data about the part publicly accessible to other parts. Promoting part features, or adding attributes, actions, or events to the interface, enables data to be shared between two parts.

View Parts List

Select **View Parts List** to open a window listing all parts that are components of the selected part. This menu option is useful if parts have been overlaid so they are not visible in the Composition Editor.

Change Name

Select **Change Name** to change the name of a part placed on the free-form surface. When you change the name of any part, you are changing the internal name of the part. This internal name appears in the status area at the bottom of the Composition Editor when you select the part. For a nonvisual part, the name also appears as a label beneath the icon for the part on the free-form surface. This menu option does not change the class name of a part.

Delete Select **Delete** to remove the selected parts.

Layout

Select **Layout** to help you place parts in your views.

Distribute
Spaces visual parts evenly within a specified window

Snap To Grid
Aligns visual parts with the nearest grid intersections

Match Size
Sizes all selected parts to match the size of the anchor part (the part selected last)

Become Primary Part
Select **Become Primary Part** to specify the primary part for your composite part. The primary part is the subpart with which most interaction will take place. It is the part whose interface is transparently exposed through the interface of the composite part. By default, the primary part of a composite part is the part that was added to the free-form surface first. For a visual composite part, this is always the Window part that VisualAge provides by default. If the pop-up menu for a part does not include **Become Primary Part**, then the part is already the primary part.

Connect
To make a connection between two parts, select **Connect** from the pop-up menu for the part or for the free-form surface. The displayed menu contains the attributes, events, and actions for the part or the application. You select the source of the connection from this menu and, after you select the target part for the connection, another menu displays for you to select the target feature. To see all attributes, events, and actions for the part or application, select **Connect → All Features**. The features in the **Connect** menu vary depending on the part selected.

Browse Connections
Select **Browse Connections** to show or hide connections to or from the part.

Show To
Displays the connections for which the part is the target

Show From
Displays the connections for which the part is the source

Show To/From
Displays the connections for which the part is either the target or source

Show All
Displays all the connections among parts in the Composition Editor

Hide To
Conceals the connections for which the part is the target

Hide From
Conceals the connections for which the part is the source

Hide To/From
Conceals the connections for which the part is either the target or source

Hide All
Conceals all the connections among parts in the Composition Editor

Reorder Conections From
Select **Reorder Connections From** to order the connections. Connections fire in the order in which they are made. However, this is not always the correct order for the connections to occur for a valid GUI application. **Reorder Connections From** enables you to order the connections in the way that works best.

Quick Form

Select **Quick Form** to create a visual part that is appropriate for displaying the value of an attribute. **Quick Form** is available for most attributes of both visual and nonvisual parts. After selecting **Quick Form**, do the following:

1. Select the attribute for which you want the corresponding quick form.
2. Move the mouse pointer to where you want to place the upper-left corner of the part. (The mouse pointer assumes the crosshair shape after you select the attribute.)
3. Click mouse button 1 to place the part.

   VisualAge automatically makes an attribute-to-attribute connection between the new part and the part from which you requested **Quick Form**. Open the settings view for the quick form to determine what other defaults were also set for you.

Tear-Off Attribute

Select **Tear-Off Attribute** to work with an attribute as if it were a standalone part. After you select **Tear-Off Attribute**, select an attribute from the displayed list and drop the tear-off attribute on the free-form surface. You can then form other connections to or from the tear-off attribute.

Create Deferred Update Part

Select **Create Deferred Update Part** to add a part that holds changes to data to the free-form surface. You can then make connections to the deferred update part. The **Create Deferred Update Part** part holds changes to data until you indicate, through its *apply* action, that the changes are to be saved.

Index