

ADOBE® ACROBAT® CONNECT™

COLLABORATION BUILDER SDK



© 2006 Adobe Systems Incorporated. All rights reserved.

Adobe® Acrobat® Connect™ Collaboration Builder SDK for Windows® and Macintosh®

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, Adobe Connect, Captivate, Flash, and FlashPaper are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Mac OS and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are the property of their respective owners.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA. Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contents

Adobe Acrobat Connect Collaboration Builder SDK	5
Audience	5
Use case examples	5
Getting started	6
Installing the Collaboration Builder SDK	6
Running your first Collaboration Builder application	7
Building Collaboration Builder applications	8
Communicating between application instances	9
Understanding the catch-up phase	9
Understanding the meeting sync state	10
Types of sync messages	10
Testing Collaboration Builder applications	11
BreezeSyncConnector component API reference	13
allowParticipantPublish()	15
breezeVersion	16
caughtUp	16
dispatchSyncMessage()	17
isArchive	18
isCaughtUp	19
isPointerOn	19
isSynced	20
isWhiteBoardOn	20
language	21
messagesCleared	21
podClosed	22
podHeight	22
podWidth	23
pointerToggle	24
role	24
roleChanged	25
sizeChanged	26
syncMessageReceived	27
syncModeChanged	28
userID	30
userLeft	30

userName	31
whiteBoardToggle	31
Index	33

Adobe Acrobat Connect Collaboration Builder SDK

The Adobe® Acrobat® Connect™ Collaboration Builder SDK lets you use Macromedia® Flash® by Adobe® to develop multiuser content for the Adobe® Acrobat® Connect™ Professional Share pod. This content can broadcast synchronization events across Connect Enterprise Server, enabling you to build applications that let people collaborate. The SDK itself is an extension package (MXP) file that includes the following:

- BreezeSyncConnector component—synchronizes a shared state in an application.
- BreezeConnectionEmulator and BreezeServicesEmulator components—allow you to test Collaboration Builder applications locally without Connect Enterprise Server.
- Source code and examples.

NOTE

Adobe Connect Enterprise Server 6 and Adobe Acrobat Connect Professional are new names for Breeze Communication Server and Breeze Meeting. The word “Breeze” is still used in the Collaboration Builder components and API.

Audience

This SDK is for developers who have a working knowledge of Flash and Flash components and want to develop collaborative Acrobat Connect Professional applications.

Use case examples

Typical applications built with the Collaboration Builder SDK have one continuous shared state, with little synchronization traffic devoted to private states. The following are a few sample applications that could be built with the Collaboration Builder SDK:

- Synchronized content navigation (for example, FlashPaper™, Adobe® Presenter, Adobe® Captivate™, brochureware)
- Dashboard applications (for example, sales dashboards with shared state and navigation)
- Collaborative applications with shared state (for example, mortgage or ROI calculators, and shared text editors)

- Multiuser games
- E-learning content

In general, any applications with shared states are good candidates for the Collaboration Builder SDK.

NOTE

The Collaboration Builder API (application programming interface) does not allow access to the Acrobat Connect Professional object model.

Getting started

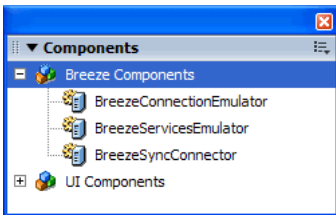
To get started using the Collaboration Builder SDK, install the MXP file and run the sample application.

Installing the Collaboration Builder SDK

To install the Collaboration Builder SDK, you need Adobe's Macromedia Extension Manager (www.adobe.com/go/em_download_en/). If you don't have the Extension Manager installed, download it and follow the installation instructions.

After you've installed the Extension Manager, click the CollaborationBuilderSDK.mxp file to install it.

When you install the CollaborationBuilderSDK.mxp file, the following three components are available in the Flash Components panel:



The BreezeSyncConnector component is the core of the SDK; it contains the API that lets the Collaboration Builder application communicate with Connect Enterprise Server. See “[BreezeSyncConnector component API reference](#)” on page 13.

The emulator components, BreezeConnectionEmulator and BreezeServicesEmulator, allow you to test applications locally, without Connect Enterprise Server. See “[Testing Collaboration Builder applications](#)” on page 11.

The component SWC files are installed in the following locations:

- Windows: Documents and Settings*username*\Local Settings\Application Data\Macromedia*Flash version*\language\Configuration\Components\Breeze Components
- Macintosh: HD/Applications/*Flash version*/Configuration/Components/Breeze Components

The component ActionScript source files are installed in the following locations:

- Windows: Documents and Settings*username*\Local Settings\Application Data\Macromedia*Flash version*\language\Configuration\Components\Breeze Components\source\
- Macintosh: HD/Applications/*Flash version*/Configuration/Components/Breeze Components/source

The BreezeServicesEmulator files and SyncSwfExample files (in both SWF and FLA formats) are installed in the following locations:

- Windows: Documents and Settings*username*\Local Settings\Application Data\Macromedia*Flash version*\language\Configuration\Components\Breeze Components\examples\
- Macintosh: HD/Applications/*Flash version*/Configuration/Components/Breeze Components/examples

Running your first Collaboration Builder application

The SyncSwfExample.flas file is a simple multiuser application that you can analyze and run on your local computer or in an Acrobat Connect Professional meeting room.

To run an application on your local computer:

1. Browse to the example files that installed with the MXP file at the following location:
 - Windows: Documents and Settings*username*\Local Settings\Application Data\Macromedia*Flash version* (Flash MX 2004 or Flash 8)\language\Configuration\Components\Breeze Components\examples\
 - Macintosh: HD/Applications/*Flash version* (Flash MX 2004 or Flash 8)/Configuration/Components/Breeze Components/examples
2. Double-click BreezeServicesEmulator.swf to open it in the stand-alone Flash Player.
3. Double-click SyncSwfExample.swf to open it in the stand-alone Flash Player.
4. Double-click SyncSwfExample.swf again to open a second instance in the stand-alone Flash Player.
5. Select items in the list boxes in the SyncSwfExample files to see the synchronization between the applications.

To run an application in an Acrobat Connect Professional meeting:

1. Close BreezeServicesEmulator.swf if it's running.
2. Open a meeting room and select the Sharing layout.

NOTE

To select the Sharing layout, you must open a room in which you are a host or presenter.

3. In the Share pod, select Documents > Select from My Computer.
4. Browse to the SyncSwfExample.swf file that was installed with the MXP file to the following location:
 - Windows: Documents and Settings*username*\Local Settings\Application Data\Macromedia*Flash version* (Flash MX 2004 or Flash 8)*language*\Configuration\Components\Breeze Components\examples\
 - Macintosh: HD/Applications/*Flash version* (Flash MX 2004 or Flash 8)/Configuration/Components/Breeze Components/examplesThe SWF file is converted and uploaded to Connect Enterprise Server.
5. Open another instance of the same meeting.
6. Choose items in the list boxes in the SyncSwfExample files to see the synchronization between the applications.

Building Collaboration Builder applications

The BreezeSyncConnector component is the core of the Collaboration Builder SDK. Methods on BreezeSyncConnector send *messages* to Connect Enterprise Server, and events dispatched from BreezeSyncConnector notify the application that messages were received or that some other state (for example role, sync, or unsync) has changed.

To build an application by using the SDK, drag a BreezeSyncConnector component to the Stage of a Flash file (the component must be on the root of the application to be detected), and give it an instance name. Use the application programming interfaces (APIs) in this document to communicate with other instances of the Collaboration Builder application. See [“BreezeSyncConnector component API reference” on page 13.](#)

Communicating between application instances

The Collaboration Builder SDK uses *sync messages* to communicate between application instances. A sync message is an object sent to each client that contains information about one aspect or property of the application; a sync message updates a client.

You send sync messages by using the `dispatchSyncMessage()` method of the `BreezeSyncConnector` component; you receive sync messages on the `Event` object of the `syncMessageReceived` event of the `BreezeSyncConnector` component. This method and event combination is the basis of communication in the SDK.

Sync messages can be any type or structure of data—they are typically, but not necessarily, objects.

Each message has a message name that you select; it's a good idea to use the name of the property the message pertains to. For example, when you synchronize the `selectedIndex` property of a `List` component, you might choose the message name "selectedIndex".

All sent messages are saved by Connect Enterprise Server to enable recording, as well as to allow users joining an existing application to catch up with the messages that were sent before they joined.

NOTE

Because of security concerns, by default, meeting participants cannot call the `dispatchSyncMessage()` method to dispatch messages through the Connect Enterprise Server. A host or presenter can call the `allowParticipantPublish()` method to allow participants to publish a specified message.

Understanding the catch-up phase

Applications that use the `BreezeSyncConnector` component go through a catch-up phase as they are loaded into a Share pod in a meeting room. This phase allows users who are new to the room to receive the accumulated state of the application. Sync messages are stored on Connect Enterprise Server as they were originally sent, and are resent privately to the new user. During this phase, while sync messages are given to the application instance (through `syncMessageReceived`) to sync the application state, the value of the `BreezeSyncConnector` component's `isCaughtUp` property is `false`. After all saved messages are received, `BreezeSyncConnector` dispatches a `caughtUp` event.

NOTE

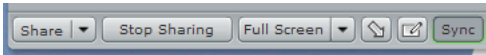
You may want your application to handle sync messages differently depending on whether or not the application is caught up to the full model.

The following events are dispatched in the following order before the catch-up phase begins:

Event	Description
<code>whiteBoardToggle</code>	Changes the <code>isWhiteBoardOn</code> property from <code>undefined</code> to <code>false</code> .
<code>pointerToggle</code>	Changes the <code>isPointerOn</code> property from <code>undefined</code> to <code>false</code> .
<code>roleChanged</code>	Changes the <code>role</code> property from <code>undefined</code> to the current user role
<code>syncModeChanged</code>	Changes the <code>isSynced</code> property from <code>undefined</code> to <code>true</code> .
<code>sizeChanged</code>	Changes the <code>podHeight</code> and <code>podWidth</code> properties from <code>undefined</code> to the height and width of the Share pod hosting the application.

Understanding the meeting sync state

The SDK allows applications to operate in either a synced or unsynced state.



Toggle the Sync button in the Share pod to change the sync state.

The `BreezeSyncConnector` component has an `isSynced` property, and a corresponding `syncModeChanged` event, that correspond to the Sync button in the Share pod. Sync messages are sent and received regardless of the sync state, but an application may choose to ignore sending or receiving outside events when the value of `isSynced` is `false`. Likewise, when not synced, the application may want to display a different user interface that allows participants to navigate where previously only presenters could (for example, as in Adobe Presenter presentations).

Additional information about the meeting that is stored and broadcast through the `BreezeSyncConnector` component include the role and user name of the current user (`role` and `userName` properties) and the size of the Share pod hosting the application (`podHeight` and `podWidth` properties).

Types of sync messages

The `BreezeSyncConnector` component supports two types of sync messages: *stateful* messages and *delta* messages. The type of message you send affects the way Connect Enterprise Server saves your messages; it also affects the way Connect Enterprise Server retrieves messages later for new users to catch up with the application state.

Stateful messages represent the entire state of a specific aspect or property. Stateful messages are the more common type of message. With stateful messages, Connect Enterprise Server saves and sends the last message of a given message name only during the catch-up phase.

Delta messages must be taken as a queue (series) to build the entire state of an application. With delta messages, Connect Enterprise Server saves all the messages of a given message name as a queue and sends the entire queue during the catch-up phase. Delta messages are practical for messages that would otherwise be very large to send.

Consider the following example: you want to synchronize two aspects of a shared text editor—the user name of a typing user, and the entire contents of the text area.

For the user name, use stateful messages named `whoIsTyping`, because only the last message is important—it contains the entire state of that property. If a new user joins during typing, the new user needs to receive only the last message.

For the contents of the text area, use delta messages named `textChanges`. As people type (imagine an increasingly long document), you can send the new text as it's being added, not the entire contents of the text area over and over again. If a new user joins during typing, Connect Enterprise Server sends the queue of `textChanges` messages, so the text area can be filled. This is a fairly uncommon case—most messages are stateful—but essential for certain situations.

To send a message as a delta, use the `BreezeSyncConnector` component's `dispatchSyncMessage()` method with the optional third parameter, `isDelta`, set to `true`. If a message queue needs to be cleared, sending one stateful sync message for that message name clears the queue of deltas, and a queue of new deltas can subsequently be added.

Testing Collaboration Builder applications

You can use the emulator components to test applications locally. You can also upload applications to Connect Enterprise Server to test applications in the Share pod of an Acrobat Connect Professional meeting room.

To test a Collaboration Builder application locally:

1. Create a FLA file with a `BreezeServicesEmulator` component in it.
Compile the FLA file, and leave the SWF file running.
2. Create a Collaboration Builder application and add a `BreezeConnectionEmulator` component and a `BreezeSyncConnector` component on the root level.
Compile the SWF file within Flash.

You see trace statements from the BreezeServicesEmulator component in the Output window acknowledging the connection.

3. Run another instance of the application SWF file (usually within the stand-alone Flash Player).

This SWF file should be in sync with the SWF application instance compiled in Flash, and trace messages should be received in the Output window for the messages being sent between the two applications.

4. When you have tested your application's logic locally, remove the BreezeConnectionEmulator component from the application's FLA file and recompile the SWF file.
5. Upload the SWF file to the Share pod of an Acrobat Connect Professional meeting room.
6. Open another instance of the same meeting room to observe the synchronization of the Share pods.

To test a Collaboration Builder application on Connect Enterprise Server:

1. If you have a previous version of the Collaboration Builder application, close the Share pod that contains it.

If you are uploading a new Collaboration Builder application, skip to step 4.

2. In the meeting room, select Pods > Organize Pods, and delete the old version of the Collaboration Builder application.
3. Close the meeting room and reenter it.
4. Upload the new Collaboration Builder application to the meeting's Share pod.
5. Open another instance of the same meeting to observe the synchronization of the Share pods.

If there is a problem with your application, return to Flash to fix the error and compile a new SWF file. Repeat this procedure.

BreezeSyncConnector component API reference

The BreezeSyncConnector component allows you to synchronize multiple SWF files loaded into Share pods in Acrobat Connect Professional meeting rooms.

Adobe recommends compiling Collaboration Builder applications with publish settings set to Flash Player 7 and ActionScript 2.0. However, if all clients are using Flash Player 6.5, you can compile with publish settings set to Flash Player 6.5.

NOTE

You cannot compile Collaboration Builder applications with publish settings set to Flash Player 8 or Flash Player 9; this means you cannot use the On2 VP6 video codec in Collaboration Builder applications.

Property summary

Properties	Description
<code>breezeVersion</code>	A numeric value that specifies the Connect Enterprise Server version number. Read-only.
<code>isArchive</code>	A Boolean value that specifies whether the application is part of an archive playback. Read-only.
<code>isCaughtUp</code>	A Boolean value that specifies whether the application has received all the sync messages recorded before a user joined (<code>true</code>) or not (<code>false</code>). Read-only.
<code>isPointerOn</code>	A Boolean value that specifies whether the pod has its pointer on (<code>true</code>) or off (<code>false</code>). Read-only.
<code>isSynced</code>	A Boolean value that specifies whether the pod has its Sync button toggled (<code>true</code>) or not (<code>false</code>). Read-only.
<code>isWhiteBoardOn</code>	A Boolean value that specifies whether the pod has its whiteboard overlay on (<code>true</code>) or off (<code>false</code>). Read-only.
<code>language</code>	A String that specifies the language of the meeting room in which the SWF file is loaded. Read-only.
<code>podHeight</code>	A number that specifies the height of the pod containing the application. Read-only.
<code>podWidth</code>	A number that specifies the width of the pod containing the application. Read-only.
<code>role</code>	A string containing one of three static properties: <code>"k_HOST"</code> , <code>"k_PRESENTER"</code> , or <code>"k_PARTICIPANT"</code> . Read-only.

Properties	Description
<code>userID</code>	A number that specifies the unique ID of the user associated with the current instance of the application. Read-only.
<code>userName</code>	A string that specifies the user name of the user associated with the current application instance. Read-only.

Method summary

Methods	Description
<code>allowParticipantPublish()</code>	Allows meeting participants to dispatch the message specified by the <code>messageName</code> parameter.
<code>dispatchSyncMessage()</code>	Sends sync messages to all other instances of the application in the given Share pod.

Event summary

Events	Description
<code>caughtUp</code>	Dispatched when the BreezeSyncConnector component finishes receiving all sync messages up to the current state of the application.
<code>messagesCleared</code>	Dispatched by a BreezeSyncConnector component only during recording playback seeking.
<code>podClosed</code>	Dispatched by a BreezeSyncConnector component when the pod containing the application is about to close.
<code>pointerToggle</code>	Dispatched by a BreezeSyncConnector component when the pointer (green arrow) is enabled or disabled.
<code>roleChanged</code>	Dispatched by a BreezeSyncConnector component when the <code>role</code> property changes.
<code>sizeChanged</code>	Dispatched by a BreezeSyncConnector component when the <code>podWidth</code> or <code>podHeight</code> properties change.
<code>syncMessageReceived</code>	Dispatched by a BreezeSyncConnector component when another instance of the application (on a different client) calls <code>dispatchSyncMessage()</code> .
<code>syncModeChanged</code>	Dispatched by a BreezeSyncConnector component when a presenter clicks the Sync button on the Share pod, and the <code>isSynced</code> property has changed.

Events	Description
<code>userLeft</code>	Dispatched by a BreezeSyncConnector component when a user leaves a meeting room.
<code>whiteBoardToggle</code>	Dispatched by a BreezeSyncConnector component when the whiteboard overlay is turned on or off over the current SWF file.

allowParticipantPublish()

Usage

```
allowParticipantPublish(messageName:String, allow:Boolean)
```

Parameters

messageName A string indicating the name of a message that meeting participants are allowed to dispatch. If the *allow* parameter is set to false, the *messageName* parameter indicates the name of a message participants are forbidden to dispatch.

allow A Boolean value indicating whether the message indicated by the *messageName* parameter can be dispatched by meeting participants (`true`) or not (`false`).

Returns

Nothing.

Description

Method; allows meeting participants to dispatch the message specified by the *messageName* parameter.

Because of security concerns, by default, meeting participants are not allowed to dispatch messages through Connect Enterprise Server. Meeting hosts and presenters can call the `allowParticipantPublish()` method to let participants send a specific message.

This method should be called in a `caughtUp` event handler.

Example

The following example allows participants to dispatch the `availableStatus` message:

```
syncConnector.allowParticipantPublish("availableStatus", true);
```

breezeVersion

Usage

```
breezeVersion:Number
```

Description

Property (read-only); a numeric value that specifies the Connect Enterprise Server version number in the format *XXX.YYY*, where *XXX* is the major version and *YYY* is the build number. For example, if the version number were 510.128, the major version would be 5.1 and the build number would be 128.

caughtUp

Usage

```
mySyncConnector.addEventListener("caughtUp", this);
```

Description

Event; dispatched by a `BreezeSyncConnector` component when it has finished receiving all sync messages up to the current state of the application. This is the point at which it is safe to send messages of your own. Until this event is fired, the `isCaughtUp` property is `false`. See [“Understanding the catch-up phase” on page 9](#).

This event has no Event object properties.

Example

This example sends the user name to all application instances when the application is caught up and keeps the list of names as a queue of delta sync messages:

```
function caughtUp(p_evt:Object){
    mySyncConnector.dispatchSyncMessage("userNames",
        mySyncConnector.userName, true);
}
```

```
mySyncConnector.addEventListener("caughtUp", this);
```

For more information about sync messages, see [“Types of sync messages” on page 10](#).

dispatchSyncMessage()

Usage

```
dispatchSyncMessage (p_msgName:String, p_msgValue [, p_isDelta:Boolean,  
p_echo:Boolean])
```

Parameters

p_msgName A string indicating the name of the message to send. Each property of the shared model should use a specific name—Connect Enterprise Server remembers either the last message of a given name (if the sync message is not a delta) or all messages of a given name (if the sync message is a delta).

p_msgValue The value of the message. Can be any data type (object is the most common). The value object is sent as a property of the Event object for the [syncMessageReceived](#) event.

p_isDelta A Boolean value that determines how the Connect Enterprise Server stores sync messages of the given name. The value `true` indicates a delta sync message, and `false` indicates a stateful sync message. The default value is `false`. This parameter is optional. If the value is `false`, Connect Enterprise Server stores only the last sent message of a given type (clearing any previous queued messages). If the value is `true`, Connect Enterprise Server appends each message of the given name to the end of a queue. See “Types of sync messages” on page 10.

p_echo A Boolean value that indicates whether the sender should have the message sent back to her. If the value is `true`, the message is sent back to the sender; if the value is `false`, everyone except the sender receives the message. This parameter is optional, and the default value is `false`.

Use the *p_echo* parameter when two users might simultaneously send a message of a certain name, and the sender wants to know the order in which her message was received. To all users who didn't send a message, the last message received is the current state, so the senders may need to know whether their message was the last to be received. If *p_echo* is set to `true`, messages are echoed back in the order they were received.

Set *p_echo* to `true` so that you don't need to manually update your local state; the [syncMessageReceived](#) handler catches the message.

Returns

Nothing.

Description

Method; sends sync messages to all other instances of the application in a given Share pod. This method cannot send private messages; everyone except the sender receives a corresponding `syncMessageReceived` event, unless the `p_echo` parameter is set to `true`, in which case the sender also receives the event. You can simulate private messages by having all users but one ignore the message after it arrives.

NOTE

Due to an application's catch-up phase, an application should wait until it has received a `caughtUp` event before it begins to send sync messages. For more information, see ["Understanding the catch-up phase" on page 9](#).

Example

In a shared `TextArea` component, suppose you want to share both a status indicator for who's typing and any changes to the text area. This function stores who's typing as a stateful message, and the text as a delta message, as follows:

```
function onKeyDown(){
  mySyncConnector.dispatchSyncMessage("whoIsTyping",
  mySyncConnector.userName);
  mySyncConnector.dispatchSyncMessage("newChar", Key.getChar(), true);
}
```

NOTE

For information about catching the corresponding events, see the [syncMessageReceived](#) example.

isArchive

Usage

`isArchive: Boolean`

Description

Property (read-only); a Boolean value that specifies whether the application is part of an archive playback (a recorded meeting is an archive). Use this property if you want to disable features or use a different display when a meeting is in playback mode.

Example

The following example disables the user interface if the application is an archive:

```
if (mySyncConnector.isArchive) {
  disableUI();
}
```

isCaughtUp

Usage

`isCaughtUp: Boolean`

Description

Property (read-only); a Boolean value that specifies whether the application has finished its catch-up phase and has received all the sync messages recorded before the user joined the meeting room. The `caughtUp` event is fired when the value of this property changes. For more information, see [“Understanding the catch-up phase” on page 9](#).

Example

The following application enables the user interface only after all messages are received:

```
if (mySyncConnector.isCaughtUp) {  
    myApp.enabled = true;  
}
```

isPointerOn

Usage

`isPointerOn: Boolean`

Description

Property (read-only); a Boolean value that specifies whether the pointer is on (`true`) or off (`false`). The `pointerToggle` event is fired when the value of this property changes.

This property is changed from `undefined` to `false` before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

Example

The following code disables clicks to the user interface while a user is using the pointer:

```
if (mySyncConnector.isPointerOn) {  
    myApp.clickEnabled = false;  
}
```

isSynced

Usage

isSynced: Boolean

Description

Property (read-only); a Boolean value that specifies whether the Share pod Sync button is toggled (`true`) or not (`false`). This doesn't affect sync messages at all—they can be sent and received as usual. However, you may want your application to send or receive sync messages, or not, based on the value of `isSynced`. See [“Understanding the meeting sync state” on page 10](#).

The `syncModeChanged` event is fired when the value of this property changes.

This property is changed from `undefined` to `true` before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

Example

The following example listens to the `syncMessageReceived` event only if the `isSynced` property is `true`:

```
function syncMessageReceived(p_evt:Object){
    if (mySyncConnector.isSynced) {
        // handlers here
    }
}
```

isWhiteBoardOn

Usage

isWhiteboardOn: Boolean

Description

Property (read-only); a Boolean value that specifies whether the whiteboard is on (`true`) or off (`false`). The `whiteBoardToggle` event is dispatched when the value of this property changes.

This property is changed from `undefined` to `false` before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

Example

The following example disables the user interface of an application when users are drawing:

```
if (mySyncConnector.isWhiteBoardOn) {
    myApp.enabled = false;
}
```

language

Usage

```
language:String
```

Description

Property (read-only); a string that specifies the language of the meeting room in which the SWF file is loaded. You can use this property to localize a pod. The following table lists the possible values:

Value	Language
"e"	English
"f"	French
"g"	German
"j"	Japanese
"k"	Korean

messagesCleared

Usage

```
mySyncConnector.addEventListener("messagesCleared", this);
```

Description

Event; dispatched by a BreezeSyncConnector component during archive seeking. The event tells the Collaboration Builder application to receive all sync messages that were recorded up to the seek point (even if those messages were previously received).

To ensure that archives play correctly during seeking, use the `messagesCleared` event to signal that the archive is starting again so that you can clear anything you did with the messages that have already played.

This event has no Event object properties.

Example

In an application with a `TextArea` component, the following function resets the `TextArea` component and the `whoIsTyping` field when the `messagesCleared` event fires:

```
function messagesCleared(p_evt:Object){
    myTextArea.text = "";
    myWhoIsTypingLabel.text = "";
}
```

```
mySyncConnector.addEventListener("messagesCleared", this);
```

podClosed

Usage

```
mySyncConnector.addEventListener("podClosed", this);
```

Description

Event; dispatched by the `BreezeSyncConnector` component when the pod containing the application is about to close, either because a meeting room layout is changing or because the Share pod was hidden. This event provides the opportunity to clean up the state of the application when the application is closing.

This event has no Event object properties.

Example

The following example clears the `whoIsTyping` field when the `podClosed` event is dispatched:

```
function podClosed(p_evt:Object){
    myWhoIsTypingLabel.text = "";
    mySyncConnector.dispatchSyncMessage("whoIsTyping", "");
}
mySyncConnector.addEventListener("podClosed", this);
```

podHeight

Usage

```
podHeight:Number
```

Description

Property (read-only); a number that specifies the height, in pixels, of the pod containing the application. The `sizeChanged` event is fired when the value of this property changes.

This property is intended to be used to lay out the application within the Share pod. See [“Understanding the meeting sync state” on page 10](#).

This property is changed from `undefined` to the current height of the Share pod before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

NOTE

The width and height of the Flash Stage have no relevance in the context of a Share pod. However, if you’re testing an application locally, `BreezeSyncConnector.podHeight` and `BreezeSyncConnector.podWidth` are always undefined so you must use the `Stage.height` and `Stage.width` properties.

Example

The following example calls a layout algorithm with the pod width and height:

```
myApp.setSize(mySyncConnector.podWidth, mySyncConnector.podHeight);
```

See also

[podWidth](#)

podWidth

Usage

`podWidth: Number`

Description

Property (read-only); a number that specifies the width of the pod containing the application in pixels. The `sizeChanged` event is fired when the value of this property changes.

This property is intended to be used to lay out the application within the Share pod. See [“Understanding the meeting sync state” on page 10](#).

This property is changed from `undefined` to the current width of the Share pod before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

NOTE

The width and height of the Flash Stage have no relevance in the context of a Share pod. However, if you’re testing an application locally, `BreezeSyncConnector.podHeight` and `BreezeSyncConnector.podWidth` are always undefined so you must use the `Stage.height` and `Stage.width` properties.

Example

The following example calls a layout algorithm with the pod width and height:

```
myApp.setSize(mySyncConnector.podWidth, mySyncConnector.podHeight);
```

See also

[podHeight](#)

pointerToggle

Usage

```
mySyncConnector.addEventListener("pointerToggle", this);
```

Description

Event; dispatched by a BreezeSyncConnector component when the pointer (green arrow) is enabled or disabled and whenever the `isPointerOn` property changes.

This event is also dispatched to set the `isPointerOn` property from `undefined` to `false` before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

This event has no Event object properties.

Example

In this example, the user interface freezes when the pointer appears:

```
function pointerToggle(p_evt:Object){  
    myUI.freeze(mySyncConnector.isPointerOn);  
}
```

```
mySyncConnector.addEventListener("pointerToggle", this);
```

role

Usage

```
role:String
```

Description

Property (read-only); a string containing one of the following three static BreezeSyncConnector properties:

```
BreezeSyncConnector.k_HOST  
BreezeSyncConnector.k_PRESENTER  
BreezeSyncConnector.k_PARTICIPANT
```

The `roleChanged` event is fired when the value of this property changes.

This property can be useful to limit or expand the user interface for users with certain roles. See [“Understanding the meeting sync state” on page 10](#).

This property is changed from `undefined` to the current user role before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

Example

The following example enables the navigation bar only if the user is a Host:

```
navBar.enabled = (mySyncConnector.role===BreezeSyncConnector.k_HOST);
```

See also

[roleChanged](#)

roleChanged

Usage

```
mySyncConnector.addEventListener("roleChanged", this);
```

Description

Event; dispatched by a `BreezeSyncConnector` component when the `role` property changes (including when the initial role is set before the catch-up phase). See [“Understanding the meeting sync state” on page 10](#).

It is possible for the `roleChanged` event to be dispatched even if the value of the `role` property hasn't changed. If your application performs complicated work in response to this event, it's best to verify that the property has changed before executing further code, as the following example shows:

```
mySyncConnector.addEventListener("roleChanged", this);
function roleChanged(p_evt:Object) {
  if (m_r===mySyncConnector.role) {
    // Make sure something really changed.
    return;
  }
  m_r = mySyncConnector.role;
  // Do routine here.
}
```

This event has no Event object properties.

Example

The following example locks down the user interface when the user's role changes to Participant:

```
function roleChanged(p_evt:Object){
  myApp.enabled = (mySyncConnector.role===BreezeSyncConnector.k_HOST);
}

mySyncConnector.addEventListener("roleChanged", this);
```

See also

[role](#)

sizeChanged

Usage

```
mySyncConnector.addEventListener("sizeChanged", this);
```

Description

Event; dispatched by a BreezeSyncConnector component when the `podWidth` or `podHeight` properties change. See [“Understanding the meeting sync state” on page 10](#).

This event is also dispatched to set the `podWidth` or `podHeight` properties from undefined to the current width and height of the Share pod before the catch-up phase. See [“Understanding the catch-up phase” on page 9](#).

It is possible for the `sizeChanged` event to be dispatched even if the values of the `podWidth` and `podHeight` properties haven't changed. If your application performs complicated work in response to this event, it's best to verify that the properties have changed before executing further code, as the following example shows:

```
mySyncConnector.addEventListener("sizeChanged", this);
function sizeChanged(p_evt:Object) {
    if (m_w==mySyncConnector.podWidth && m_h==mySyncConnector.podHeight) {
        // Make sure something really changed.
        return;
    }
    m_w = mySyncConnector.podWidth;
    m_h = mySyncConnector.podHeight;
    // Do complicated layout routine here.
}
```

This event has no Event object properties.

Example

In the following example, the application resizes when the size of the pod changes:

```
function sizeChanged(p_evt:Object){
    myApp.setSize(mySyncConnector.podWidth, mySyncConnector.podHeight);
}

mySyncConnector.addEventListener("sizeChanged", this);
```

syncMessageReceived

Usage

```
mySyncConnector.addEventListener("syncMessageReceived", this);
```

Description

Event; dispatched by a `BreezeSyncConnector` component when another instance of the application (on a different client) calls the `dispatchSyncMessage()` method. The `messageName` and `messageValue` are sent as properties of the Event object.

The `BreezeSyncConnector` component also dispatches this event during the catch-up phase, to bring the current application instance up to date with the messages Connect Enterprise Server previously recorded during the meeting. During the catch-up phase, each recorded message is dispatched in the order in which it was recorded until the client is brought up to date. For more details on how messages are recorded by Connect Enterprise Server, see [“Building Collaboration Builder applications” on page 8](#).

This event has the following Event object properties:

Name	Type	Description
<code>messageName</code>	String	The name of the message sent by the <code>dispatchSyncMessage()</code> method.
<code>messageValue</code>	Any	The value of the message sent by the <code>dispatchSyncMessage()</code> method.
<code>userID</code>	Number	The user ID of the application client instance that sent the message. (This property is not usually necessary, but is potentially useful.)
<code>isDelta</code>	Boolean	Whether or not the message was sent as a delta. (For information about delta messages, see “Types of sync messages” on page 10 .)

Example

In the following example, suppose you have a shared `TextArea` component and would like to share both a status indicator for who's typing and any changes to the text. The function receives the `whoIsTyping` message as a stateful message, and receives the text as a delta message, `newChar`.

```
function syncMessageReceived(p_evt:Object){
  if (p_evt.messageName=="whoIsTyping") {
    myWhoIsTypingLabel.text = p_evt.messageValue;
  } else if (p_evt.messageName=="newChar") {
    myTextArea.text += p_evt.messageValue;
  }
}

mySyncConnector.addEventListener("syncMessageReceived", this);
```

NOTE

For details on sending the corresponding message, see the [dispatchSyncMessage\(\)](#) example.

syncModeChanged

Usage

```
mySyncConnector.addEventListener("syncModeChanged", this);
```

Description

Event; dispatched by a `BreezeSyncConnector` component when a presenter or host clicks the Sync button on the Share pod and the `isSynced` property has changed. See [“Understanding the meeting sync state” on page 10](#).

This event is also dispatched to set the `isSynced` property from undefined to true before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

This event has the following Event object properties:

Name	Type	Description
<code>didISync</code>	Boolean	If the user of the current application instance was the user who clicked the button, this is <code>true</code> . If any other user clicks the button, the property is <code>false</code> . You can use this property to resynchronize all other users to the user who synced. For example, if the <code>syncModeChanged</code> event is invoked with the <code>didISync</code> property set to <code>true</code> , the <code>dispatchSyncMessage()</code> method could be called with the current state of the user's application to bring other users up to sync.

Example

In the following example, when the application is unsynchronized, it stops sending sync messages when a user types in a `TextArea` component; each user's `TextArea` component now contains its own text. When a user resynchronizes the application, the application sends all the text from that user's `TextArea` component so that all users receive the changes.

```
function syncModeChanged(p_evt:Object){
  if (mySyncConnector.isSynced && p_evt.didISync) {

    // Some message that tells clients to clear their TextAreas.
    mySyncConnector.dispatchSyncMessage("clearAll");

    // Note, this isn't a delta, clearing the saved queue.
    mySyncConnector.dispatchSyncMessage("newChar", myTextArea.text);

  } else if (!mySyncConnector.isSynced) {
    // Set some variable that makes sure we don't send any messages
    // as we type
  }
}

mySyncConnector.addEventListener("syncModeChanged", this);
```

userID

Usage

userID: Number

Description

Property (read-only); a number that specifies the unique ID of the user associated with the current instance of the application. This property could be used to send private messages (messages that are broadcast to everyone, but filtered by user ID so that only the intended user actually reacts to it), or for any other case that requires unique identification of a user. This property is available after the `caughtUp` event is dispatched. See [“Understanding the catch-up phase” on page 9](#).

Example

The following example reacts only to a sync message intended for this user:

```
function syncMessageReceived(p_evt:Object){
    // For this example, I'm sending a destinationID as part of the
    // sync message. If my ID matches, then the message is meant for me.
    if (p_evt.messageValue.destinationID == mySyncConnector.userID) {
        // Do something.
    }
}
```

userLeft

Usage

```
mySyncConnector.addEventListener("userLeft", this);
```

Description

Event; dispatched by a `BreezeSyncConnector` component when a remote user leaves a meeting room. You can use this event when a user leaves a meeting room and you want to clear a state that was allocated to them.

This event has the following Event object properties:

Name	Type	Description
userID	Number	The ID of the user who left the meeting room.

Example

The following example deletes a user from the user list when the user leaves the application:

```
function userLeft(p_evt:Object){
    myUserList.removeUserID(p_evt.userID);
}
```

```
mySyncConnector.addEventListener("userLeft", this);
```

userName

Usage

```
userName:String
```

Description

Property (read-only); a string that specifies the user name of the user associated with the current application instance. This property is useful for display purposes. This property is available after the `caughtUp` event is dispatched. See [“Understanding the catch-up phase” on page 9](#).

Example

The following example sends a delta message that tells everyone a user has joined the application:

```
mySyncConnector.dispatchSyncMessage("userJoined",
    {userName:mySyncConnector.userName, userID:mySyncConnector.userID},
    true);
```

whiteBoardToggle

Usage

```
mySyncConnector.addEventListener("whiteBoardToggle", this);
```

Description

Event; dispatched by a `BreezeSyncConnector` component when the whiteboard overlay is turned on or off over the current SWF file and whenever the `isWhiteBoardOn` property changes.

This event is also dispatched to set the `isWhiteBoardOn` property from `undefined` to `false` before the catch-up phase of an application. See [“Understanding the catch-up phase” on page 9](#).

This event has no Event object properties.

Example

In this example, the user interface freezes when the whiteboard is turned on:

```
function whiteBoardToggle(p_evt:Object){
    myUI.freeze(mySyncConnector.isWhiteBoardOn);
}
mySyncConnector.addEventListener("whiteBoardToggle", this);
```

Index

A

- allowParticipantPublish() method 15
- API, BreezeSyncConnector component 13
- application
 - building 8
 - sample 7
 - testing 11
- audience 5

B

- BreezeConnectionEmulator component
 - installing 6
 - testing locally with 11
- BreezeServicesEmulator component
 - installing 6
 - testing locally with 11
- BreezeSyncConnector component
 - about 8
 - API reference 13
 - installing 6
- breezeVersion property 16
- building 8

C

- catch-up phase 9
- caughtUp event 16
- CollaborationBuilderSDK.mxp file 6
- components
 - BreezeConnectionEmulator 6
 - BreezeServicesEmulator 6
 - BreezeSyncConnector 8, 13
- Components panel 6

D

- dispatchSyncMessage() method 17

E

- events, summary 14
- Extension Manager. *See* Macromedia Extension Manager

F

- Flash Player, supported version 13

G

- getting started 6

I

- installing 6
- isArchive property 18
- isCaughtUp property 19
- isPointerOn property 19
- isSynced property 20
- isWhiteBoardOn property 20

L

- language property 21

M

Macromedia Extension Manager 6
messagesCleared event 21
methods, summary 14
MXP file 6

O

Output window 11

P

podClosed event 22
podHeight property 22
podWidth property 23
pointerToggle event 24
properties, summary 13
publishing 13

R

role property 24
roleChanged event 25

S

sample application 7
sizeChanged event 26
state, shared 5
sync messages
 about 9
 catch-up phase 9
 delta 11
 stateful 11
syncMessageReceived event 27
syncModeChanged event 28

T

testing 11
trace statements 11

U

use cases 5
userID property 30
userLeft event 30
userName property 31

W

whiteBoardToggle event 31