

## Трюки и приемы использования BitmapData во Flex 2

Автор: [Andrew Trice](#)

Перевод: [Андрей Горбатов](#)

[Оригинал статьи](#)

### Что такое Bitmap API?

Bitmap API появилась в восьмой версии плеера. Каждый визуальный объект в swf отображается как bitmap данные (которые кэшируются), вместо ежекадровой прорисовки. Только объекты, отмеченные как "dirty", будут прорисованы заново. Таким образом уменьшается количество затрачиваемых ресурсов и увеличивается производительность. Некоторую информацию о Bitmap API можно найти на:

- [http://www.adobe.com/devnet/flash/articles/bitmap\\_caching.html](http://www.adobe.com/devnet/flash/articles/bitmap_caching.html)
- [http://www.adobe.com/devnet/flash/articles/image\\_api\\_03.html](http://www.adobe.com/devnet/flash/articles/image_api_03.html)

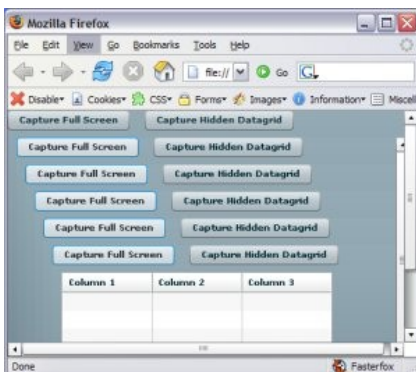
Но зачем это нужно во Flex приложении? Дело в том, что это относится ко всем Flex компонентам вашего приложения. Они прорисовываются и затем кэшируются внутри Flash плеера. Это не только увеличивает производительность, но и дает нам доступ к bitmap данным соответствующего объекта. Ниже приведена небольшая, но мощная функция, позволяющая захватывать BitmapData ЛЮБОГО визуального Flex компонента. Использован класс [UIComponent](#) в качестве параметра, так как он является основным для всех компонентов.

```
private function getUIComponentBitmapData( target : UIComponent ) : BitmapData
{
    var bd : BitmapData = new BitmapData( target.width, target.height );
    var m : Matrix = new Matrix();
    bd.draw( target, m );
    return bd;
}
```

Что же с этим делать? А вот что:

1. Различные эффекты с изображениями
2. Зеркальное отображение UI компонентов
3. Расшаривание приложений
4. Экспорт скриншотов

В первом примере получают BitmapData определенного компонента и передаются компоненту Image. Этот пример можно найти на [JAM](#).



```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">

<mx:Script>
<![CDATA[

import mx.core.UIComponent;

private function captureFullScreen() : void
{
var bd : BitmapData = getBitmapData( UIComponent( mx.core.Application.application ) );
targetImage.source = new Bitmap( bd );
}

private function captureHiddenDatagrid() : void
{
var bd : BitmapData = getBitmapData( UIComponent( hiddenDg ) );
targetImage.source = new Bitmap( bd );
}

private function getBitmapData( target : UIComponent ) : BitmapData
{
var bd : BitmapData = new BitmapData( target.width, target.height );
var m : Matrix = new Matrix();
bd.draw( target, m );
return bd;
}

]]>
</mx:Script>

<mx:Button
id="captureButton"
label="Capture Full Screen"
click="captureFullScreen()" />

<mx:Button
id="captureButton2"
label="Capture Hidden Datagrid"
click="captureHiddenDatagrid()"
x="153"/>

<mx:Image
id="targetImage"
x="10"
y="30"/>

<mx:DataGrid
x="99"
y="64"
id="hiddenDg"
visible="false">
<mx:columns>
<mx:DataGridColumn headerText="Column 1" dataField="col1"/>
<mx:DataGridColumn headerText="Column 2" dataField="col2"/>
<mx:DataGridColumn headerText="Column 3" dataField="col3"/>
</mx:columns>
</mx:DataGrid>

</mx:Application>

```

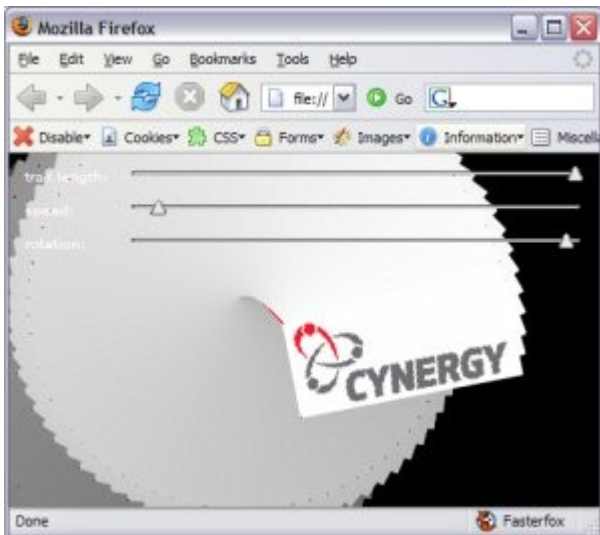
[Запустить приложение](#)

[Исходник](#)

[Загрузить исходник](#)

## Эффекты с изображениями

Использование bitmap данных позволяет создавать интересные эффекты. В данном случае имеется изображение в точке 0,0 размером с приложение. Прозрачность изображения составляет 90%. Каждое событие "ENTER\_FRAME" создается снимок приложения и отображается в изображении. Получается эффект следа. Код краток, но эффективен.



*Заметка: Жрет много ресурсов.*

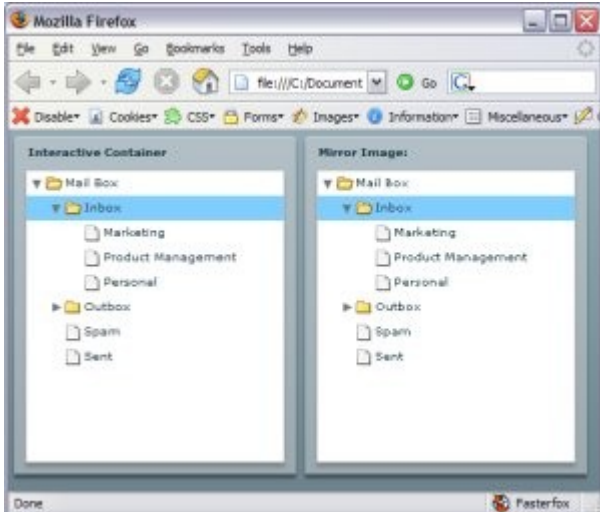
[Запустить приложение](#)

[Исходник](#)

[Загрузить исходник](#)

### **Зеркальное отображение**

В следующем примере получают BitmapData виджета Tree левой части и отображаются каждые 100 миллисекунд в правой части.

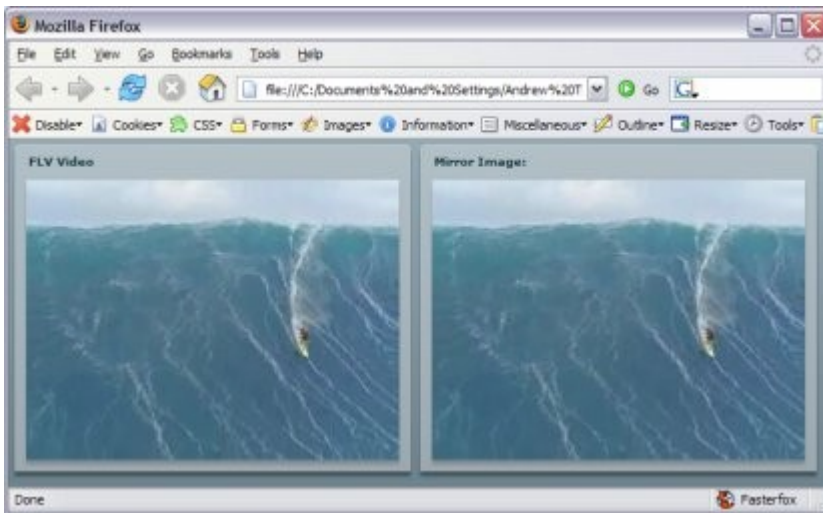


[Запустить приложение](#)

[Исходник](#)

[Загрузить исходник](#)

В следующем примере происходит зеркальное отображение проигрываемого flv файла.



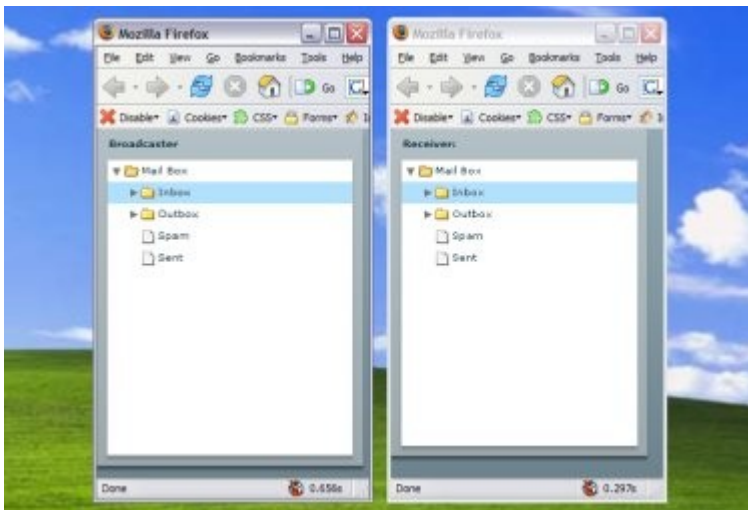
[Запустить приложение](#)

[Исходник](#)

[Загрузить исходник](#)

### Расшаривание приложений

В следующем примере происходит расшаривание данных между двумя браузерами с помощью local shard objects. Этот пример работает на одной машине, но имейте в виду, что можно использовать remote shared object и будет работать на разных машинах! То есть можно будет распределять изображения с сервера на клиентские машины без использования Flash Media Server.



[Посмотреть Демо](#)

[Запустить транслятора](#)

[Запустить получателя](#)

[Исходник](#)

[Загрузить исходник](#)

*Заметка: В данном примере были использованы свои методы вместо `getPixels` и `setPixels`, так как в последних имеет место ошибка Flash плеера, при которой возвращается "End Of File" при использовании этих методов с элементами, содержащимися в памяти плеера. Adobe уже предупрежден. Не был использован объект `BitmapData` object с local shared object, потому что плеер не может десериализовать его обратно в `BitmapData` при чтении*

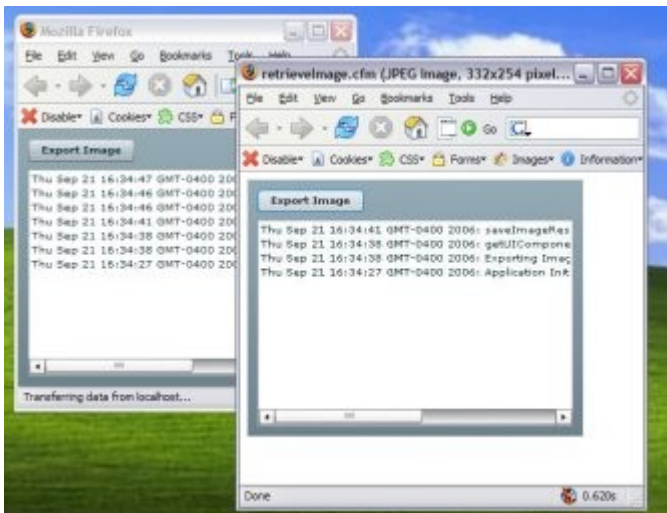
из local shared object. Вместо этого он был сохранен в LSO как ByteArray.

## Экспорт скриншотов

Хотели ли вы когда-нибудь экспортировать текущее состояние приложения в картинку? С помощью данного подхода можно захватывать состояние приложения и сохранять его в jpg, который может быть загружен пользователем. Использовать это можно в следующих случаях:

- Создание скриншотов ошибок
- Сохранение изображений приложения!
- Расшаривание приложений

В следующем примере захватываются BitmapData, конвертируются в JPG ByteArray с помощью класса JPEGEncoder, который находится в [ActionScript 3 corelib](#). При наступлении события ResultEvent, происходит запрос с сервера и изображение загружается в новом окне. В качестве бэкэнда использован ColdFusion.



[Посмотреть Демо](#)

[Запустить приложение](#)

[Посмотреть часть исходника](#)

[Загрузить часть исходника](#)