Adobe AIR Local Data Storage Options (With Emphasis on Using Embedded SQL Databases)

Greg Hamer
President
Halcyon Solutions, Inc.
greg@halcyonsolutions.net

Session Outline

- OVERVIEW
- AIR LOCAL STORAGE OPTIONS
  - flash.net.SharedObject
  - flash.filesystem.File and FileStream
  - flash.data.EncryptedLocalStore
  - flash.data.SQLConnection and SQLStatement
- USING AIR LOCAL SQL DATABASE
- Q&A

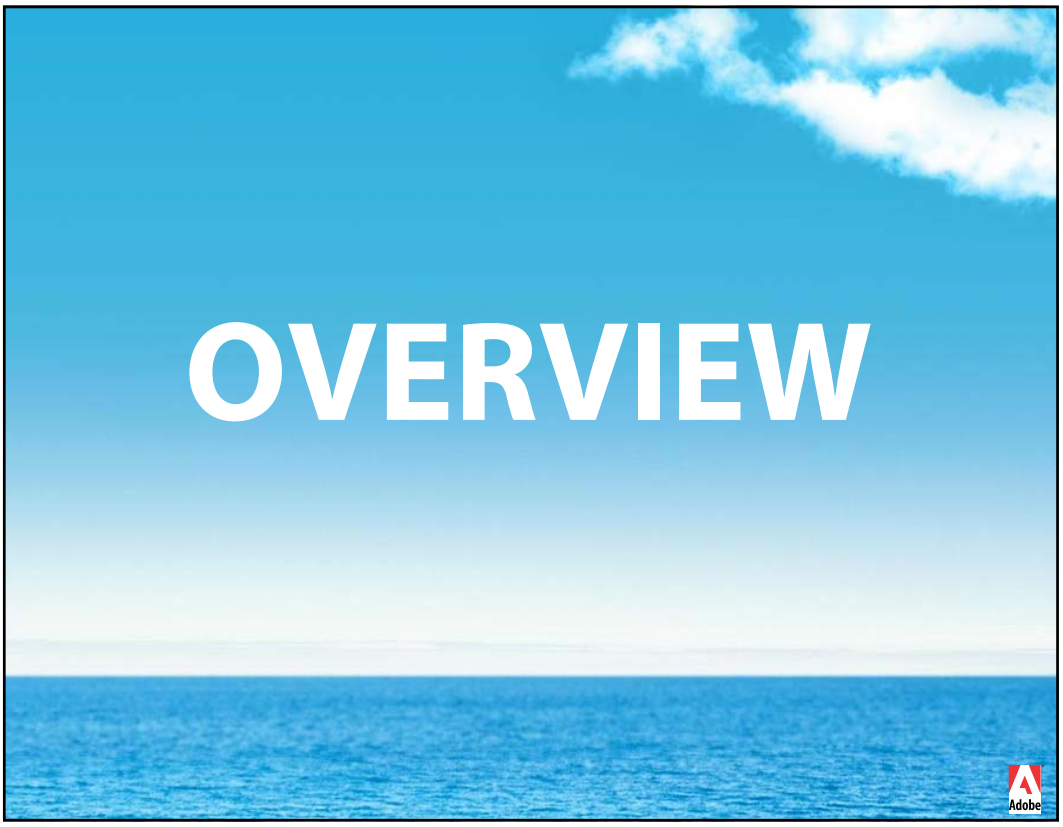
Note: Links in presentation online at:
http://halcyonsolutions.net/preso/08/airsql/

Introduction

- Greg Hamer
  greg@halcyonsolutions.net
- Web application architect and developer, entrepreneur and educator
  - Flex 3, Flash (since MX), Flash Media Server, Flash Video, ColdFusion, Java, SQL (MySQL, Microsoft, Sybase), HTML.
  - Adobe Certified Instructor (all of the above)
  - Active Developer (all of the above)

Who This Session is Targeted At

- Application architects considering adoption of Adobe AIR
- Developers and designers seeking to take advantage of the opportunities that AIR is opening up for them!
  - Flex
  - Flash
  - HTML/AJAX



AIR APIs – What Do They Mean?

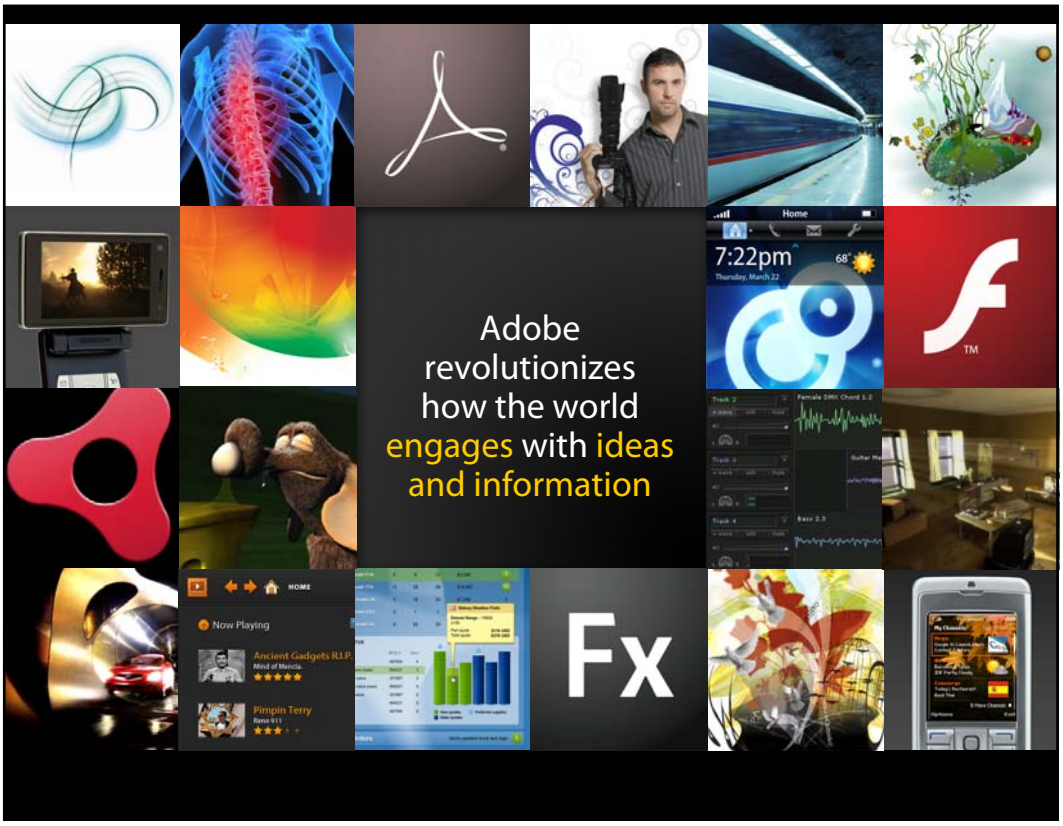
- APIs for Flex, Flash and HTML/AJAX developers!
  - New, cross operating system runtime: AIR
    - Today: Windows and Mac
    - Tomorrow: Linux and mobile
- "Microsoft is trying to bring the .Net community to the Web. We are really focused on bringing the larger Web community to the desktop. It's two different approaches. It's not a head-on thing--it's just two groups of developers," [Kevin] Lynch said. "Our bet is on the Web."
  Source: http://www.news.com/8301-10784_3-9789007-7.html

ADOBE® AIR™

Adobe AIR enables web developers to use existing technologies to build and deploy rich Internet applications on the desktop.

Building AIR Applications

- Leverage existing tools for designers and developers
- Flex, Aptana, Dreamweaver, Flash, and more



The ABCs of AIR

Always There, **A**nywhere

Persistent rich internet applications on the desktop or devices across any major operating system

Branded Experiences

Fully branded experiences not constrained by the browser or desktop

Convenient Desktop Functionality

Drag & drop, copy & paste, task manager and notifications

Data Access

Sort, save, see and send data locally or on the server

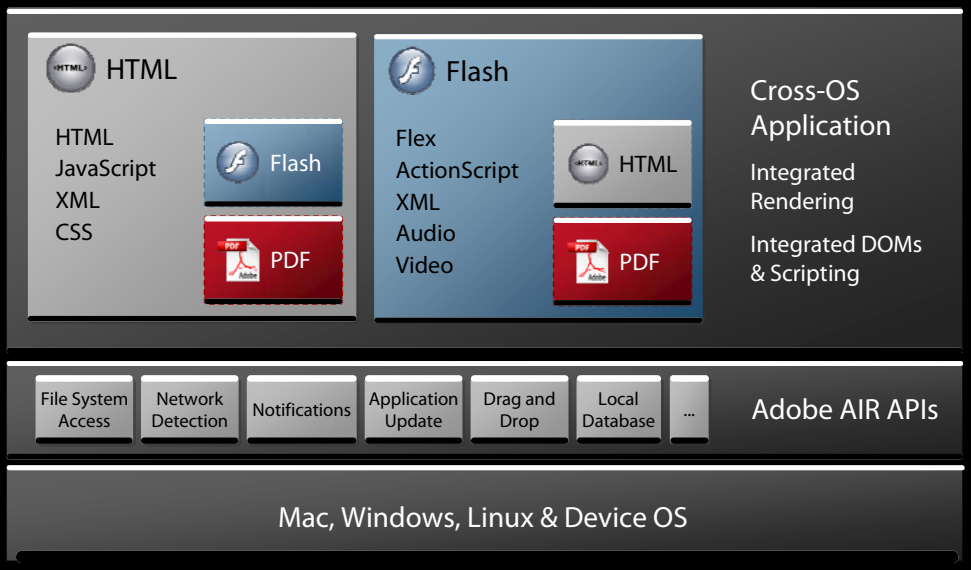
Efficient Development and Delivery

Use existing web skills and technologies to deliver new apps faster…seamlessly deploy and update

ADOBE AIR

Desktop RIA

Adobe AIR Application Stack



AIR Positioning vis a vis Web Browsers

- ***AIR builds upon browser capabilities and technologies by adding valuable desktop presence, persistence, more flexible user interface control, convenient desktop functionality and local data access.***
- **Browser Strengths**
- Content-based sites, community-oriented apps, widely distributed applications
- Collecting usage data and rapidly improving an application or content
- UI-oriented development with sophisticated design tools

- **Browser Weaknesses**
- Dependant on the browser being open and an internet connection being available
- **Security sandbox** restricts desktop access, other domains, and requires browser chrome

AIR Positioning vis a vis Dedicated Desktop Applications

- ***AIR can be used to replace many dedicated desktop applications providing convenient desktop / local system integration and data access.***

Desktop Strengths

- Desktop real estate is valuable – persistent customer connection…tend to be used more often and not dependent on being connected
- Can take advantage of the power of the OS, local files, local memory and storage

Desktop Weaknesses

- Market shifting from desktop to web applications.
- Apps are operating system specific - Desktop clients are expensive to build, update, deploy and manage
- Lacks tools for creating compelling user experiences

Two-Day Workshops

- Adobe AIR w/ Flex, Flash & HTML/AJAX
  - April 26 & 27 (Sat/Sun) – Las Vegas
  - May 6 & 7 (Tue/Wed) – Toronto
- RichMediaInstitute.com

Adobe

Storage Models

- AIR Applications can read and write data using four storage mechanisms
  - Local Shared Objects
  - Encrypted Local Store
  - File System
  - Embedded SQL Database

Adobe

Storage Models -- Comparison

Classes Packages	Perf Read	Perf Write	Runtime Supporting	Strong Suits	Storage Location
`SharedObject` flash.net	Slow	Fast	AIR or Flash Player	-- Same syntax in both AIR and Flash Player -- No storage limitation in AIR	[1]
`File` / **`FileStream`** flash.filesystem	Varies *May be Worst depending on location of data in file*	Fast	AIR only	-- Documents, images, etc (Note: using SQL still an option) -- Flex includes UI components: `FileSystemList`, `FileSystemTree`, `FileSystemDataGrid`	[2]
`EncryptedLocalStore` flash.data	Slow	Worst	AIR only	-- Encrypted = Secure; privacy	[3]
`SQLConnection` / **`SQLStatement`** flash.data	Best	Varies *(indexes slow writes)*	AIR only	-- SQL Standard, significantly same syntax as enterprise SQL (e.g. Oracle, mySQL, etc.) -- Local replication of server side enterprise data can duplicate existing server side SQL table schemas	[2]

[1] E.g. on Windows C:\Documents and Settings\admin\Application Data\{application ID}\Local Store\#SharedObjects\LocalStorage_Demo.swf\
C:\Documents and Settings\admin\Application Data\LocalStorage-Demo\Local Store\#SharedObjects\LocalStorage_Demo.swf\

[2] Configurable. By convention often in File.applicationStorageDirectory.

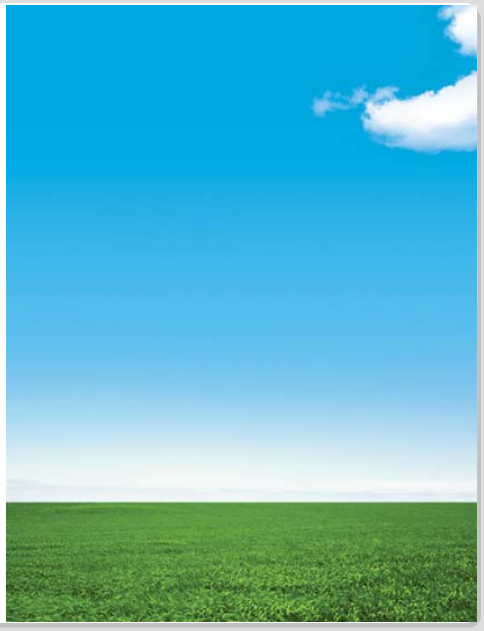
[3] Encrypted local store data is put in a subdirectory of the user's application data directory; the subdirectory path is Adobe/AIR/ELS/ followed by the application ID. E.g. on Windows: C:\Documents and Settings\admin\Application Data\Adobe\AIR\ELS\{application ID}

Storage Models -- Performance

	Create data: 5,000 records	Save data *(writes to disk)*	Get item 4,999
`SharedObject`	48(ms)	172(ms)	94(ms)
`File / FileStream`	63(ms)	411(ms)	345(ms)
`EncryptedLocalStore`	92(ms)	**885,724(ms)**	158(ms)
`SQLConnection / SQLStatement`	49(ms)	1,321(ms)	**0(ms)**

- Performance using AIR_LocalStorage_Demo tool
- AIR_LocalStorage_Demo created by Jason Williams
  - Adobe Sr. Computer Scientist on AIR team
  - Lead on implementing SQLite in AIR
- DEMO

Local Shared Objects



Adobe

Local Shared Objects

- In package flash.net.*
- SharedObject used to serialize memory resident data structures
- Synchronous mode
- Uses AMF3/AMF0 for serialization
- No default size limit
- Monolithic storage model

Adobe

Example

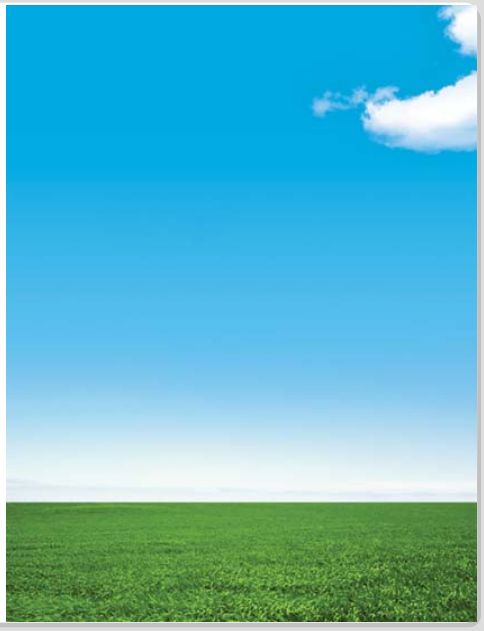
```
var lso:SharedObject =
SharedObject.getLocal("myLocalData");

var items:Array = [];
items.push({name:"Bob", id:"555-55-5555"});
items.push({name:"Sam", id:"555-66-5555"});

lso.data.items = items;
lso.flush();
```


Adobe

Encrypted Local Store



Adobe

Encrypted Local Store

- In package flash.filesystem.*
- Used to store sensitive data
- Synchronous mode
- All data is serialized using ByteArray
- Uses AES-CBC 128-bit encryption based on App ID and user

Example

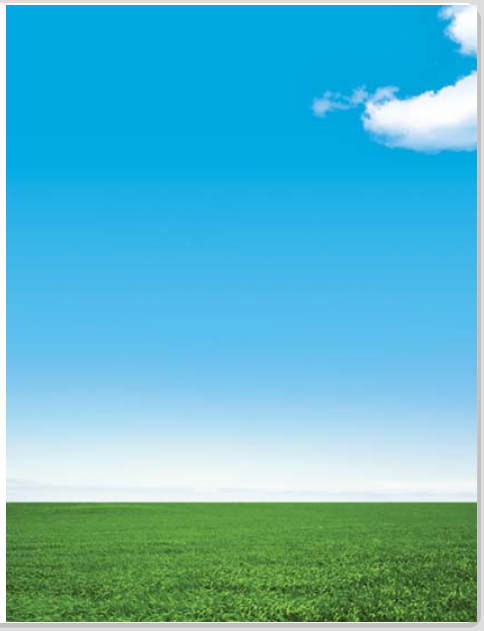
```
var items:Array = [];
items.push({name:"Bob", id:"555-55-5555"});
items.push({name:"Sam", id:"555-66-5555"});
var ba:ByteArray = new ByteArray();
ba.writeObject(items);

// writing secure data
EncryptedLocalStore.setItem("contacts", ba);

// reading secure data out
ba = EncryptedLocalStore.getItem("contacts");
var obj:Object = ba.readObject();

// removing secure data
EncryptedLocalStore.removeItem("contacts");
```


File System API



Adobe

File System

- In package flash.filesystem.*
- Random access to file system data.
- Asynchronous and Synchronous modes
- Two main components
  - **`File`**
  - **`FileStream`**

*A 10 point footnote can go here, if necessary

Adobe

File System Components

- **`File`**
  - Represents a path to a file or directory
  - OS-independent path manipulation
  - Well-known locations (Desktop, Documents, temp files, etc.)
  - Listing Directories
  - Move, copy, delete, and create
- **`FileStream`**
  - Provides IO (Binary, Object, or Text data encoded AMF3/AMF0)

Example

```
import flash.filesystem.*;

var file:File = File.documentsDirectory;
file = file.resolvePath("AIR Test/customers.dat");
var stream:FileStream = new FileStream();
stream.open(store, FileMode.WRITE);
for (var i:int=0; i<items.length; i++)
{
  stream.writeObject(items[i]);
}
stream.close();
```


Cataloging Information

- Application Folders:
  - **`File.applicationResourceDirectory`** – read-only directory containing compiled application assets
  - **`File.applicationStorageDirectory`** – an automatically created, applicatino-specific read/write directory intended for persistent, offline file storage. Can be accessed by all users on that single computer.
- User Folders
  - **`File.userDirectory`** -- user's desktop directory
    - Windows: C:\Documents and Settings\[user]\
    - Mac: /Users/[user]/
  - **`File.desktopDirectory`** -- user's desktop directory
  - **`File.documentsDirectory`**
    - Windows: C:\Documents and Settings\[user]\My Documents
    - Mac: /Users/[user]/Documents

Alteration

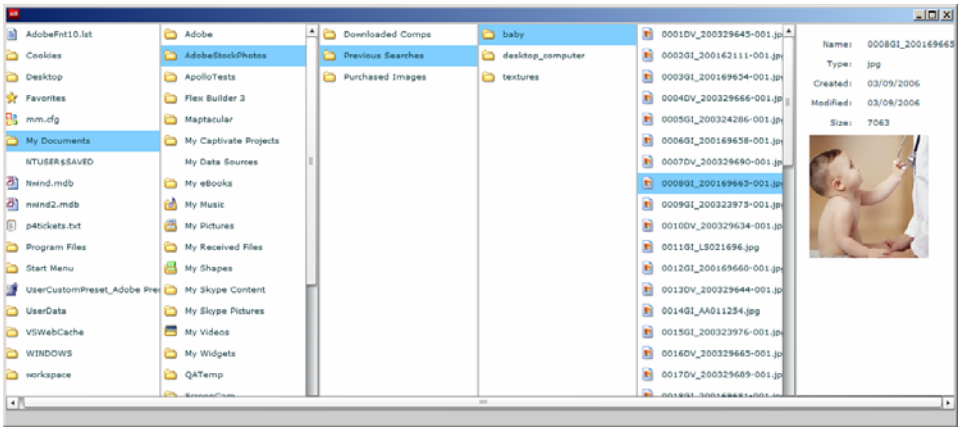
- Copying and Moving
  - File.copyTo()
  - File.moveTo()
- Deletion
  - File.deleteFile()
  - File.deleteDirectory()
  - File.moveToTrash()

Miscellaneous

- Upload and download
  - File.upload()
  - File.download()
  - File.send();
- Native Browse Dialogs
  - File.browse()
  - File.browseForDirectory()
  - File.browseForOpen()
  - File.browseForMultipleOpen()
  - File.browseForSave()

Adobe

Demo

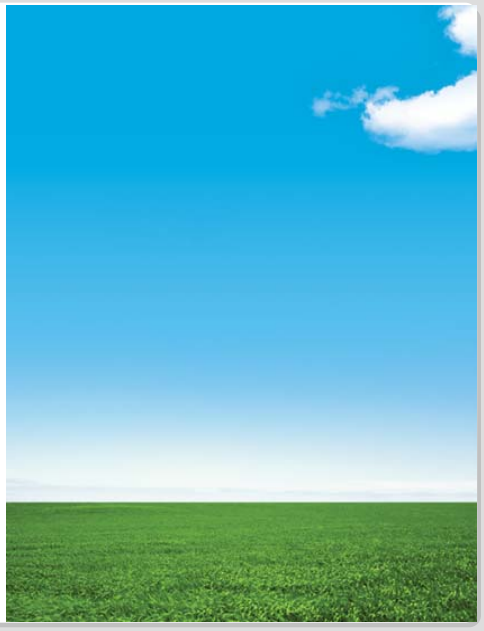


Adobe

USING AIR LOCAL SQL DATABASE

Adobe

Embedded SQL DB



Adobe

Benefits of the embedded DB

- **Self Contained** – no external dependencies
- **Single File** – each database is stored completely within a single file. Files are sharable across supported operating systems.
- **Zero Setup** - no configuration or administration is required. There is nothing to be installed, no server process, no need for an administrator to create a database, users, and accounts. There are no configuration files.
- **Transactional** – each operation that modifies the data will be atomic, consistent, isolated, and durable (ACID).
- **Large capacity** – each database file can grow to a theoretical limit of over 2 TB (2^41 Bytes) depending on operating system limits.

Adobe

Embedded SQL DB

- In package flash.data.*
- Relational data storage
- Asynchronous and Synchronous modes
- Two main classes
  - `SQLConnection`
  - `SQLStatement`

*A 10 point footnote can go here, if necessary

Adobe

SQLite: What is it?

- Public domain
  - http://sqlite.org/
- Support for UTF-8 and UTF-16
- File format is cross-platform
- Compact Library
  - ~250kB loaded, can be reduced to 180kB
  - Ideal for "constrained devices"
- Supports most of ANSI-SQL 92
  - Only a limited subset of ALTER TABLE is supported — to drop or alter columns, you must drop the entire table and recreate it)
- Supports views, temporary tables
- Can be used by an AIR app regardless of whether a network connection is available because the database runs and data files are stored locally

SQLite: What it's not

- No stored procedures
- No enforcing of data type constraints
- No foreign key constraints
- Primary keys must be integers

SQLite in AIR: What it's not

- A multiuser, enterprise database
- Many issues of multiuser, enterprise database are absent:
  - High Concurrency
  - High-volume
  - Very large datasets

Adobe

SQLite: What it's not

- Some idiosyncratic conventions:
  - Manifest typing instead of typed columns
  - No size constraints on columns
  - Primary Keys — integers only
- Can perform operations synchronously or asynchronously
- No errors when addressing non-existent columns

Adobe

DOCUMENTATION

- SQL support in local databases
  - http://livedocs.adobe.com/flex/3/langref/localDatabaseSQLSupport.html
  - http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/localDatabaseSQLSupport.html
- FLEX
  - dev_guide_flex_air1.pdf | Chapter 21
  - http://livedocs.adobe.com/flex/3/html/SQL_01.html
- HTML/AJAX
  - http://livedocs.adobe.com/air/1/devappshtml/SQL_01.html
- FLASH
  - http://livedocs.adobe.com/air/1/devappsflash/SQL_01.html

Storage Classes

- The storage classes that are compatible with SQL92
  - **NULL** - The value is a NULL value (null in AS3)
  - **INTEGER** - The value is a signed integer (int in AS3)
  - **REAL** - The value is a floating point value (Number in AS3)
  - **TEXT** - The value is a text string, stored using UTF16 (String in AS3)
  - **BLOB** - The value is a blob of data (returned as specified in AS3)

Affinity

- Declaring a data type on a column determines the column's affinity.
- Affinity determines the recommended storage class.
- Supported ActionScript affinities
  - **Boolean**
  - **Date**
  - **int**
  - XML
  - XMLList
  - Object
  - uint
  - Number
  - String

Example

```
"CREATE TABLE customers (id int PRIMARY KEY, name TEXT,
                         birth_day Date, active Boolean,
                         address BLOB);";


"CREATE TABLE customers (id int PRIMARY KEY, name String,
                         birth_day Date, active Boolean,
                         address Object);";
```


Embedded SQL DB

- **SQLConnection**
  - Establishes connection state
  - Manages creation and configuration
  - Transactions
  - Schema access
  - Provides background thread for processing statements in async mode
- **SQLStatement**
  - Provides Create, Read, Update and Delete (CRUD)
  - Parameters
  - Paging
  - Custom result row data types

Example (Read)

```
import flash.filesystem.*;
import flash.data.*;

var file:File = File.documentsDirectory;
file = file.resolve("AIR Test/test.db");
var conn:SQLConnection = new SQLConnection(true);
conn.open(file);

var stmt:SQLStatement = new SQLStatement();
stmt.sqlConnection = conn;
stmt.itemClass = Customer;
stmt.text = "SELECT * FROM customers WHERE name LIKE :nm; ";
stmt.parameters[":nm"] = custNameInput.text + "%";
stmt.execute();
customerGrid.dataProvider = stmt.getResult().data;
```

Connecting to Data

- **`SQLConnection.open()`** – establishes the connection
  - In-memory
  - Persistent
- **`SQLConnection.openAsync()`** – establishes the connection
  - In-memory
  - Persistent
- **`SQLConnection.attach()`** – adds a database to the current connection.
  - In-memory
  - Persistent

Adobe

Example (attach)

```
import flash.filesystem.*;
import flash.data.*;

var conn:SQLConnection = new SQLConnection();
conn.open();
var file:File = File.documentsDirectory;
file = file.resolve("AIR Test/another.db");

conn.attach("cust_db", file);
var stmt:SQLStatement = new SQLStatement();
stmt.sqlConnection = conn;
stmt.itemClass = Customer;
stmt.text = "SELECT * FROM cust_db.customers WHERE name LIKE :nm; ";
stmt.parameters[":nm"] = custNameInput.text + "%";
stmt.execute();
customerGrid.dataProvider = stmt.getResult().data;
```


Adobe

Getting the results

- **`SQLStatement.getResult()`**
  - LIFO queue of SQLResult objects.
- **`SQLResult`**
  - **`SQLResult.data`** – array of objects that contain each row of the result.
  - **`SQLResult.complete`** – indicates if this is a partial result, i.e. if a page size was specified and not all of the data was returned
  - **`SQLResult.lastInsertRowID`** – the value of the ROWID for the most recent row inserted.
  - **`SQLResult.rowsAffected`** – number of rows that were changed based on an INSERT, UPDATE, or DELETE operation. Doesn't include trigger operations.

Putting data on disk

- Writing data is a six step process
  - 1. Acquire a SHARED lock on the db.
  - 2. Acquire a RESERVED lock on the db – only one connection can hold a reserved lock for any process.
  - 3. The in-memory roll back journal is updated.
  - 4. The contents of the roll back journal are then written to the disk surface.
  - 5. Acquire an EXCLUSIVE lock on the db
  - 6. Write all modifications for the db to the disk surface

Transactions

- Multiple mutation operations can be applied within one write operation.
- Enables undo on failure.
- Managed by **`SQLConnection`**.
  - **`SQLConnection.begin()`**
    - Deferred - doesn't acquire any locks until the first write operation (allows read access when locked)
    - Immediate - acquires locks immediately (allows read access when locked).
    - Exclusive - acquires locks immediately and prevents other processes from reading
  - **`SQLConnection.commit()`**
  - **`SQLConnection.rollback()`**

Using query parameters

- **`SQLStatement`** supports using query parameters
- Parameters are an associative array to which you add values for the parameters specified in the SQL statement's **`text`** property
- The array keys are the names of the parameters
- If an unnamed parameter is specified in the statement **`text`**, its key is the index of the parameter
- Within the **`text`** of a SQL statement, a parameter is indicated with one of the following characters: "?", ":", or "@"
  - "?" is for indexed parameters
  - ":" and "@" are for named parameters

Example

```
import flash.filesystem.*;
import flash.data.*;

//...
stmt.text = "INSERT INTO customers VALUES(:id, :nm :date);
";
conn.begin();
for(var i:int=0; i<data.length; i++)
{
   stmt.parameters[":nm"] = data[i].name;
   stmt.parameters[":date"] = data[i].date;
   stmt.parameters[":id"] = data[i].id;
   stmt.execute();
}
conn.commit();
```


Adobe

Introspection

- Access to table/view, column, index, and trigger information, including the SQL used to create the entity.
- Selective loading
- Works against all connected databases
- SQLConnection.loadSchema()
- SQLConnection.getSchemaResult() – LIFO queue of SQLSchemaResult objects.

Adobe

Introspection

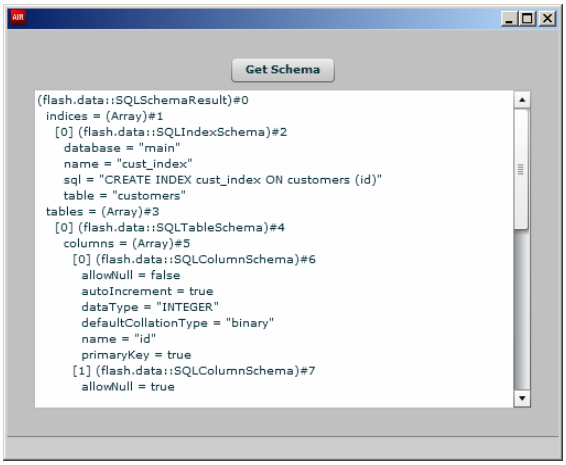
- SQLSchemaResult
  - tables – SQLTableSchema *(name, database, SQL, and columns)*
    - SQLColumnSchema *(name, primary key, allows null values, auto increment, declared data type, default collation type)*
  - views – SQLViewSchema *(name, database, SQL, and columns)*
  - triggers – SQLTriggerSchema (*name, database, table, and SQL)*
  - indexes – SQLIndexSchema *(name, database, table, and SQL)*

Example

```
var file:File = File.documentsDirectory;
file = file.resolve("AIR Test/test.db");
var conn:SQLConnection = new SQLConnection(true);
conn.open(file);

con.loadSchema();
var result:SQLSchemaResult = con.getSchemaResult();
// look at tables
for (var i:int=0; i<result.tables.length; i++)
{
   var tableSchema:SQLTableSchema = result.tables[i];
   for (var j:int=0; j<tableSchema.columns.length; j++)
      displayColumnInfo(tableSchema.columns[j]);
}
```


Demo



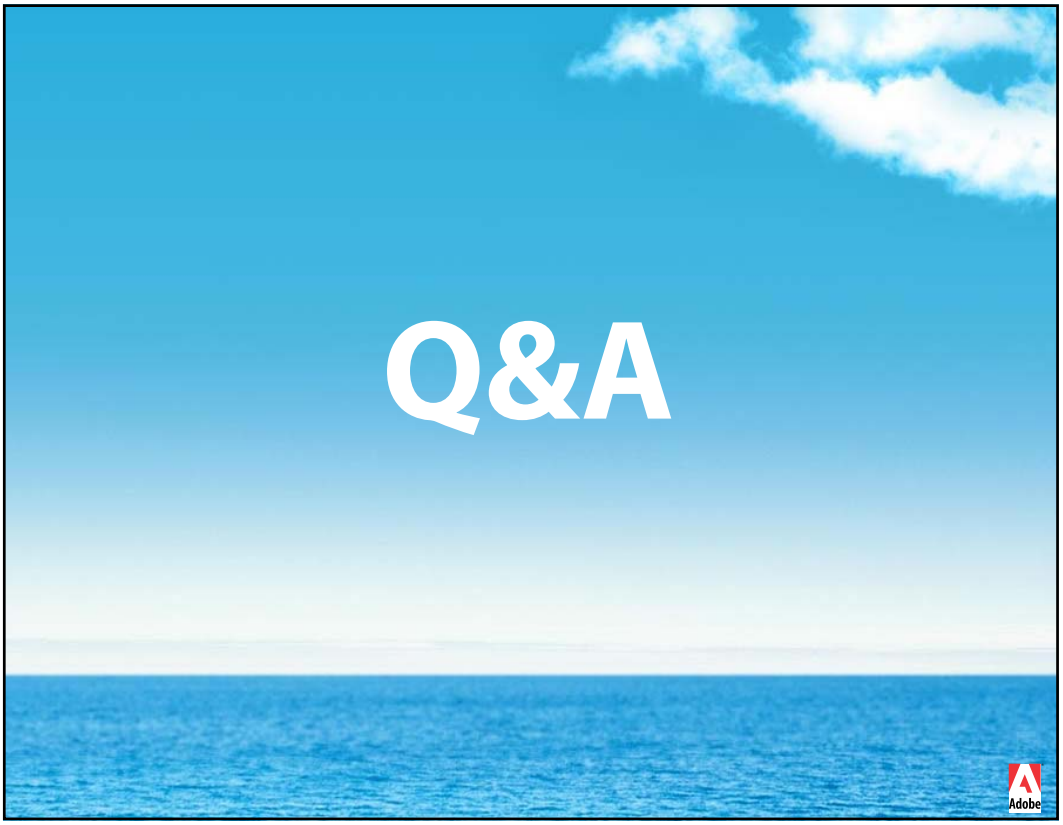
Adobe

OTHER

Adobe

Other

- Adobe Evangelist Christophe Coenraets:
  - Offline Synchronization using AIR and LiveCycle Data Services
    http://coenraets.org/blog/2007/10/offline-synchronization-using-air-and-livecycle-data-services/
  - Simple ORM Framework for AIR via Annotating ActionScript Classes with Custom Metadata
    http://coenraets.org/blog/2007/10/annotating-actionscript-classes-with-custom-metadata-simple-orm-framework-for-air/



Session Link & Email Contact

- Note: Links in presentation online at: http://halcyonsolutions.net/preso/08/airsql/
- Email: greg@halcyonsolutions.net

Adobe