2 Development of multimedia applications

Literature: Derek Franklin, Jobe Makar: Flash MX 2004 actionscript, Macromedia Press 2004 (Chapters 17 and 18)

Playing Video from Animations

- Embedding video information into animation
  - Leads to very large files (SWF files in the case of Flash)
- External video clips:
  - Editable separately with specialized software
  - Progressive download: play during loading
  - Video played at its own frame rate, not at the rate of the animation
- Support for external video in Flash (MX 2004):
  - FLV (Flash Video) format
  - Converters from most well-known video formats to FLV exist
  - Special *Media Components* for easy integration of video
    - » MediaDisplay
    - » MediaController
    - » MediaPlayback (= MediaDisplay + MediaController)
  - Media component can also play back MP3 audio

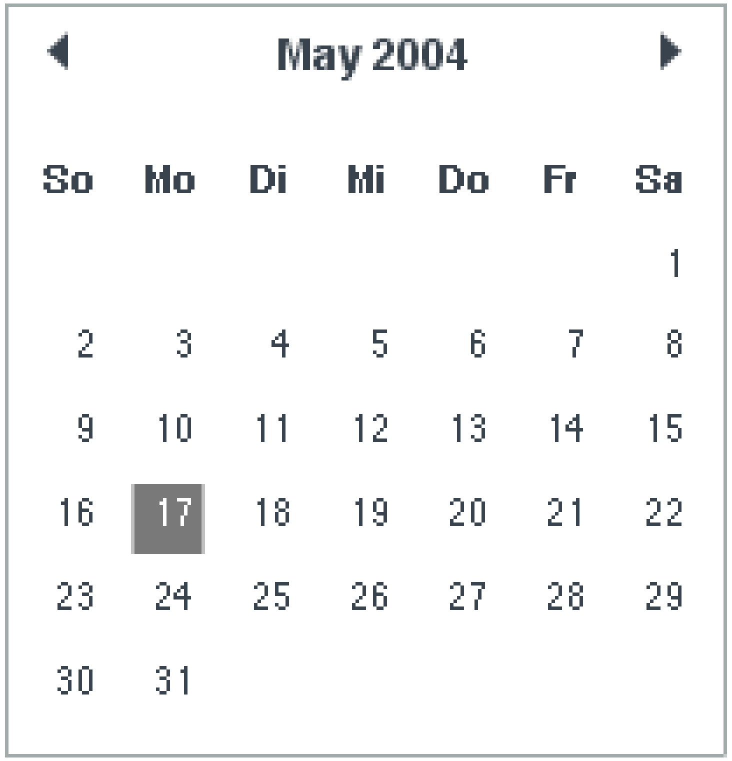
Flash Components

- *Software component:* „A ***software component*** is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.“

  ECOOP 1996, Workshop on Component-oriented Programming
- *Flash component:* A reusable unit of Flash design and ActionScript programming with clearly specified parameters and methods. A Flash component encapsulates a ready-made solution that can be incorporated into third-party Flash applications.
- Components delivered with Flash (MX 2004, examples):
  - User Interface components:
    - » Button, CheckBox, ComboBox, DataGrid, DateChooser, Label, ProgressBar, ScrollPane. TextArea, TextInput, Window, ...
  - Data components:
    - » DataHolder, DataSet, WebServviceConnector, ...
  - Manager:
    - » PopUpManager, Depth Manager, ...
  - Media Components ...

Example Flash Component: Date Chooser

- Layout and basic behaviour pre-defined
- Component inspector allows customization, e.g.
  - Definition of string representation for days, months
  - Disabled days (not chosable)
  - Start day of week
- API allows dynamic ActionScript-based adaptation
  - E.g. setting selected date
- Components generate events



So	Mo	Di	Mi	Do	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

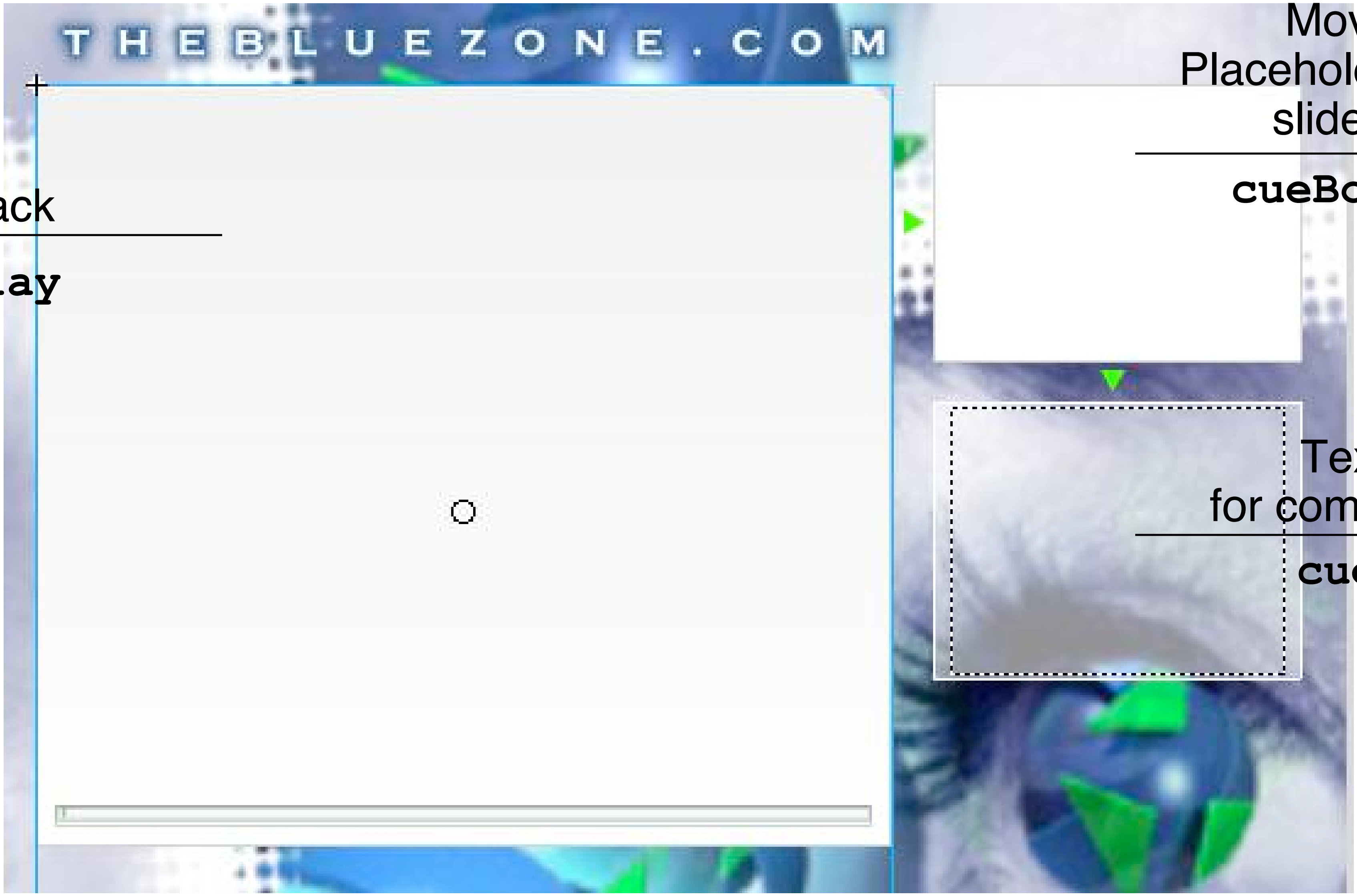
Events Generated by Media Components

- Various events are reported by Media Components to the surrounding application for flexible reaction:
  – Adjustments like change of volume
  – Media events like reaching end of media
  – User-defined events when reaching specific positions *(cue events)*

- Reaction to media events requires *Listener* objects, e.g.

```
var myListener:Object = new Object();
myListener.volume = function() {
  // actions to react on volume change
}
myMediaComponent.addEventListener(“volume”, myListener);
```

Example: Video with Event-Triggered Animation



Media Playback

`display`

MovieClip Placeholder for slide show

`cueBox_mc`

Text field for comments

`cue_txt`

Step 1: Setting Component Parameters

- Component parameters can be set
  - With the component inspector (authoring tool)
  - By script commands

```
display.autoPlay = true;
  // start playing immediately
display.activePlayControl = true;
  // display playback button as active
display.controllerPolicy = "on";
  // controls always visible
display.totalTime = 60;
  // total runtime to be used in playback progress bar
```

Step 2: Add Required Event Listeners

- Example:
  - Listener for “complete” event (i.e. end of video)
  - Automatically invokes a browser window with a given URL

```
var displayListener:Object = new Object();
displayListener.complete = function(){
  getURL("http://www.thebluezone.com");
}
display.addEventListener("complete", displayListener);
```

Step 3: Load External File

- Both filename and file extension are specified, since also MP3 files can be played
- Playback started
  - Automatically via auto-play parameter setting (as in the example)
  - When user presses “play” button in controller
  - Controlled by script

```
display.setMedia("bluezone.flv", "FLV");
```

Cue Points

- A *cue point* marks a specific point in time during media playback.
  - Cue points can be defined independently of the movie (in ActionScript)
  - When reaching a cue point, an event is fired which can be handled by ActionScript.

```
display.addCuePoint("0", 1);
display.addCuePoint("1", 8);
display.addCuePoint("2", 14);
display.addCuePoint("3", 31);
display.addCuePoint("4", 35);
display.addCuePoint("5", 53);
display.addCuePoint("6", 56);
display.addEventListener("cuePoint", displayListener);
displayListener.cuePoint = function(eventObj:Object){
  var index = Number(eventObj.target.name);
  loadMovie("cue" + index + ".jpg", "cueBox_mc");
  cue_txt.text = cueTextArray[index];
}
```

Cue Points in the Example

- Names of cue points chosen in a way such that conversion to number gives an index
- Two arrays of information to be displayed in the two extra windows
  - Still pictures
  - Text information

cue2.jpg

“Fluffy is crammed into dial-up pipe”

cueTextArray[2]

Flash Pattern: Names and Numbers

- **Problem:** Indexing and computing an index requires numbers to identify information instances. Storage in files and symbol identifiers require strings to identify information instances.
- **Solution:**
  – When a string is required to be used as an index: Choose a string representing a number and convert to number when required with function **`Number()`**
  – When a number is required to be used as a string: Compute an appropriate String by concatenating a base string with the number. Choose file names and identifiers appropriately.
- **Known Uses:**
  – String-to-Number: Cue point names in above example
  – Number-to-String: File names for Cue*X* pictures in above example; Sound names in Basketball example

2 Development of multimedia applications

Elements of XP and their Applicability to Multimedia Authoring

• The Planning Game	applicable directly
• Small releases	applicable directly
• Metaphor	applicable in adapted way
• Simple design	What is simple design in e.g. Flash?
• Testing	How to automate tests for Flash?
• Refactoring	applicable in adapted way
• Pair programming	applicable directly
• Collective code ownership	applicable directly
• Continuous integration	applicable directly
• 40-hour week	applicable directly
• On-site customer	applicable directly
• Coding standards	applicable / which standards?

Metaphor and Multimedia Authoring

- Metaphor practice in XP:
  - To find a single metaphor which represents the system’s functionality in real world’s terms.
- Multimedia applications (e.g. with Flash/ActionScript):
  - Key step of design is to find a convincing graphical representation
  - If representation and interaction designed right:
    - » Almost tangible, immersive user experience
    - » Excellent metaphor: Possibilities for interaction help in understanding and presenting the system functionality
- Advice:
  - Take graphical design seriously.
  - **Start** from graphical design, **refine** into sophisticated program.
  - Use **authoring tool as the bridge** between worlds of graphical design and programming.

Simple Design: Trade-Offs in Multimedia Authoring

- The following trade-offs are obvious with Flash, but exist in some form in any multimedia authoring tool.
- **Design vs. Behaviour** trade-off
  - Shall we work out the (graphical/sound) design first, or shall we try to understand the full interaction structure first?
- **Scripting/architecting** trade-off
  - Shall we simply start to spread scripting code over the animation symbols, or shall we try to design a scripting architecture first?
  - Examples for scripting architectures:
    - » MVC: Account example
    - » Only global code: Sound, movie examples
    - » All code in external classes: Drag examples
- **Instance/schema** trade-off
  - Shall we simply assign behaviour to instances, or shall we bother to define generic behaviour and customization for the instances?

Extreme Multimedia Authoring: What is it?

- Here are some suggestions (to be verified during projects):
- Start from the graphical design, but integrate simple behaviour soon.
  - Study alternatives to find the most natural way of representing the system.
  - Refine later (design and behaviour)
- Use the ***simplest possible scripting*** approach first.
  - Keeping code on the main timeline is not object-oriented, but often helpful for experiments
- Refactor ***when necessary***:
  - Complex business logic with multiple views: Introduce MVC architecture
  - Multiple instances of a symbol with generic behaviour:
    - » Attach code to symbols and their instances
    - » Consider usage of linked ActionScript classes
  - Refactoring is easy: Code snippets can be moved around
- Create a testing infrastructure and archive tests
  - Always test full functionality after refactoring
- Work in ***short evolution cycles!***

2 Development of multimedia applications

Testing ActionScript Classes

- Principle:
  - Test cases written as ActionScript code
  - Conventions (and test framework) for systematic execution of tests
- Ideal case:
  - Test framework available: Usually derivations of xUnit (e.g. JUnit)
  - *ASUnit* test framework for ActionScript available on the Web, but based on AS 1 :-(
- Workaround:
  - Build your own simple test infrastructure (for a base version, see below)
- Limitations:
  - Does work only with pure ActionScript classes
  - Calling event handlers is doubtful (causality of events not assured)
  - Graphical input and output cannot be used
- Consequence:
  - Usable mainly in case of MVC architecture or similar architectures

Simple Test Infrastructure: Superclass Test

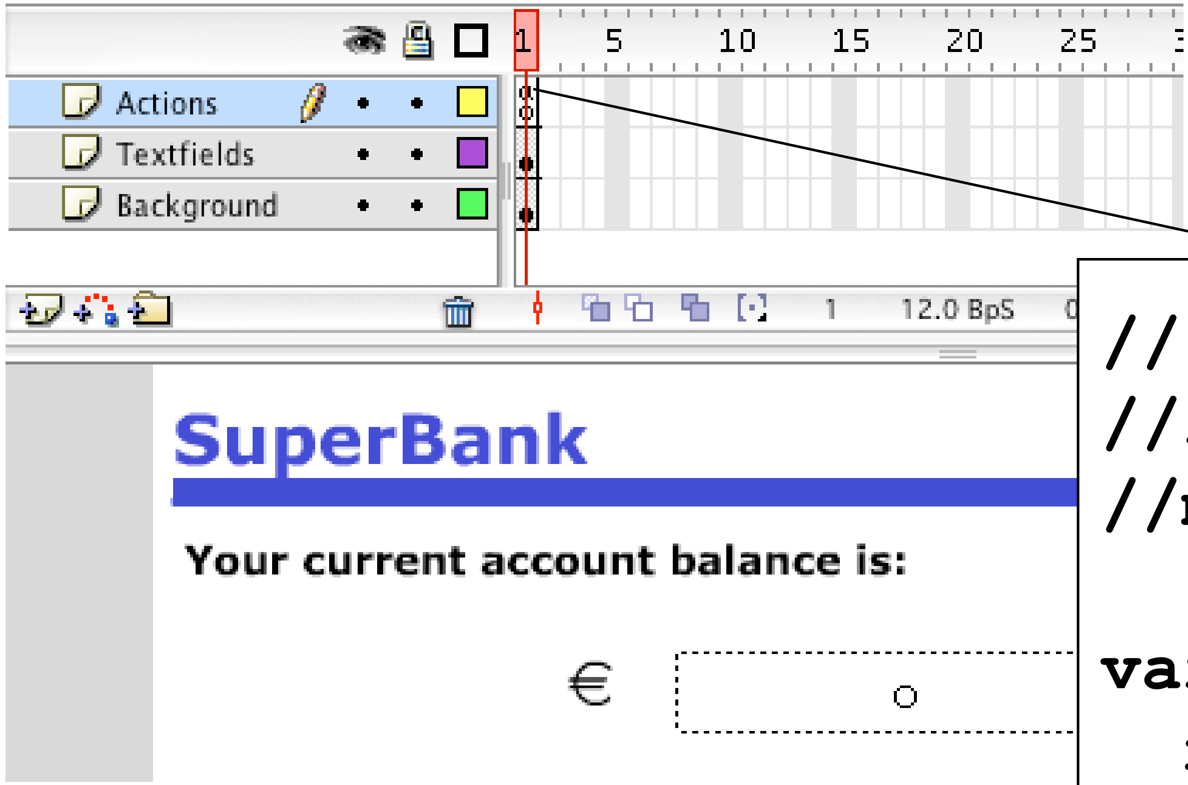
```
class Test {

  var success:Boolean = true;

  function setUp() {};      //abstract
  function classTest() {};  //abstract

  function assertEqualNumber(x:Number, y:Number) {
      if (not(x == y))
           success = false;
  }

  function runTest() {
      setUp();
      classTest();
      if (success)
           trace("Test successful!");
      else
           trace("Sorry, test failed.")
  }

}
```

Design pattern applied: “Template Method” (Gamma et al.)

Simple Test: Class AccountTest

```
class AccountTest extends Test {

  var acc1, acc2:Account;

  function setUp() {
      acc1 = new Account(123);
      acc2 = new Account(234);
  }

  function classTest() {
      var amount:Number = 100;
      acc1.debit(amount);
      acc2.credit(amount);
      acc1.credit(amount);
      acc2.debit(amount);
      assertEqualNumber(acc1.getSaldo(),0);
      assertEqualNumber(acc2.getSaldo(),0);
  }

}
```

Simple Test: Make It Mandatory



```
// First of all,
//let's test the
//model classes

var at =
   new AccountTest();
at.runTest();

...
```

Automated Test of User Interaction?

- For most Flash/ActionScript applications:
  - Pure ActionScript tests not sufficient
  - What can we do?

- The simplest (but not worst!) solution:
  - Use *discipline* to run tests manually
  - Keep archive of test descriptions to reproduce tests

- More automatic approach:
  - Use software for user interaction scripting
  - E.g.
    - » AutoIt (www.autoitscript.com) for Windows
    - » AppleScript (www.apple.com/applescript) for MacOS
      - Only suitable for special applications, Flash Player currently excluded

2 Development of multimedia applications

Literature: Derek Franklin, Jobe Makar: Flash MX 2004 actionscript, Macromedia Press 2004 (Chapters 17 and 18)

Loading Variables

- Method loadVariables() to load data from external source
  - Load can take place from local file or from server over network (http)
- Special class **`LoadVars`** to maintain name/value pairs loaded from external source
  - Signals event when loaded completely
  - Example:

```
var container:LoadVars = new LoadVars();
container.load(...);
```

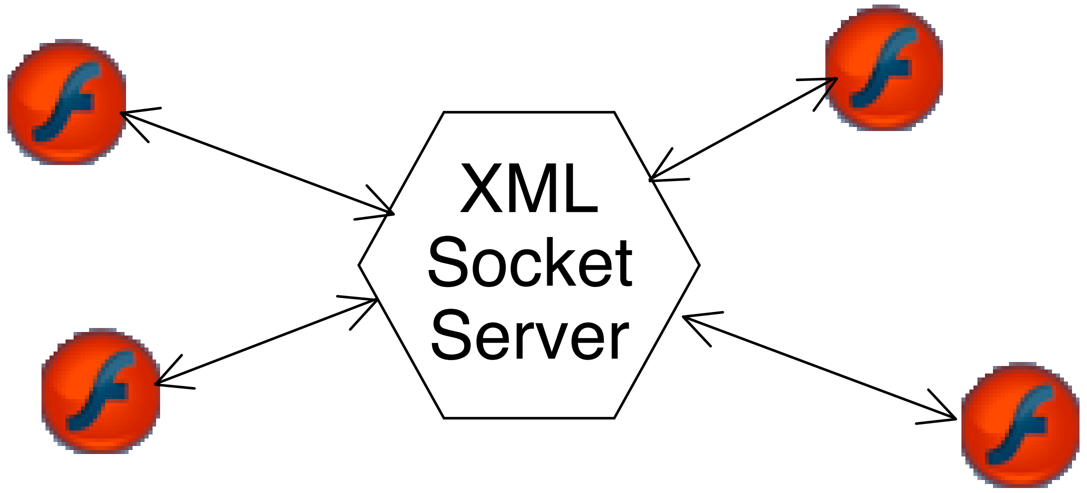
- String (URL) representation of loaded data (“form url-encoded”)
  - Example:

```
name=michael&age=23&phone=113344
```

XML Files in Flash

- A standard way for storing semi-structured data is XML
  - Built-in support in Flash
- Class **`XML`** for objects representing XML information
  - API for reading and manipulating tree representation: **`attributes(), childNodes(), hasChildNodes(), removeNode(), createElement(), insertBefore(), ...`**
- Typical methods for loading data:
  - **`load()  //load from a URL`**
  - **`send()  //send to a URL`**
  - **`sendAndLoad()`**

XML Socket Server



- Simple, lightweight, ***low-latency*** solution to realize communication among various Flash applications
  - Suitable for chat rooms and simple multi-user games
  - Free of license for small user numbers
- Information can be sent or received at any time over a socket connection

XMLSocket Class

- `var server:XMLSocket = new XMLSocket();`
- Connecting to a server
  - `server.connect(hostName, port);`
  - There must be special software running on the server machine, e.g. the free Java-based *ElectroServer* software (www.electrotank.com)
- Sending to the server:
  - `Server.send("text");`
- Closing the connection:
  - `Server.close();`
- Events:
  - `onXML: Fired when receiving XML data`
  - `onConnect: Fires when connect operation ends`
  - `onClose: Fires when connection is lost`
- Proprietary high-level API to XMLSockets for *ElectroServer*:
  - Send and receive XML-based data without using XML syntax

Selected ElectroServer Methods (1)

```
class ElectroServer extends XMLSocket {
  public static function getInstance():ElectroServer;
  function getIP():String;
  function getPort():Number;
  function setIP(tempIP:String):
  function setPort(tempPort:Number);
  function send(action:String, parameters:String);
  function sendPublicMessage
      (message:String, variables:Object);
  function sendPrivateMessage
      (message:String, users:Array, variables:Object);
  function login(tempUsername:String, tempPassword:String) {
  function changeRoomDetail(detail:String, value);
  function deleteRoomVariable(name:String);
  function kick(name:String, reason:String);
  function ban(name:String, reason:String, expires);
  ...
```

Selected ElectroServer Methods (2)

```
  ...
  function createUserVariable(name:String, value:String);
  function updateUserVariable(name:String, value:String);
  function deleteUserVariable(name:String, value:String);
  function createRoomVariable(ob:Object);
  function getZone();
  function createGameRoom(roomOb:Object) {
  function joinGame(room:String, password:String,
   type:String, zone:String) {
  function adminLogin
   (tempUsername:String, tempPassword:String) {
  ...
}
```