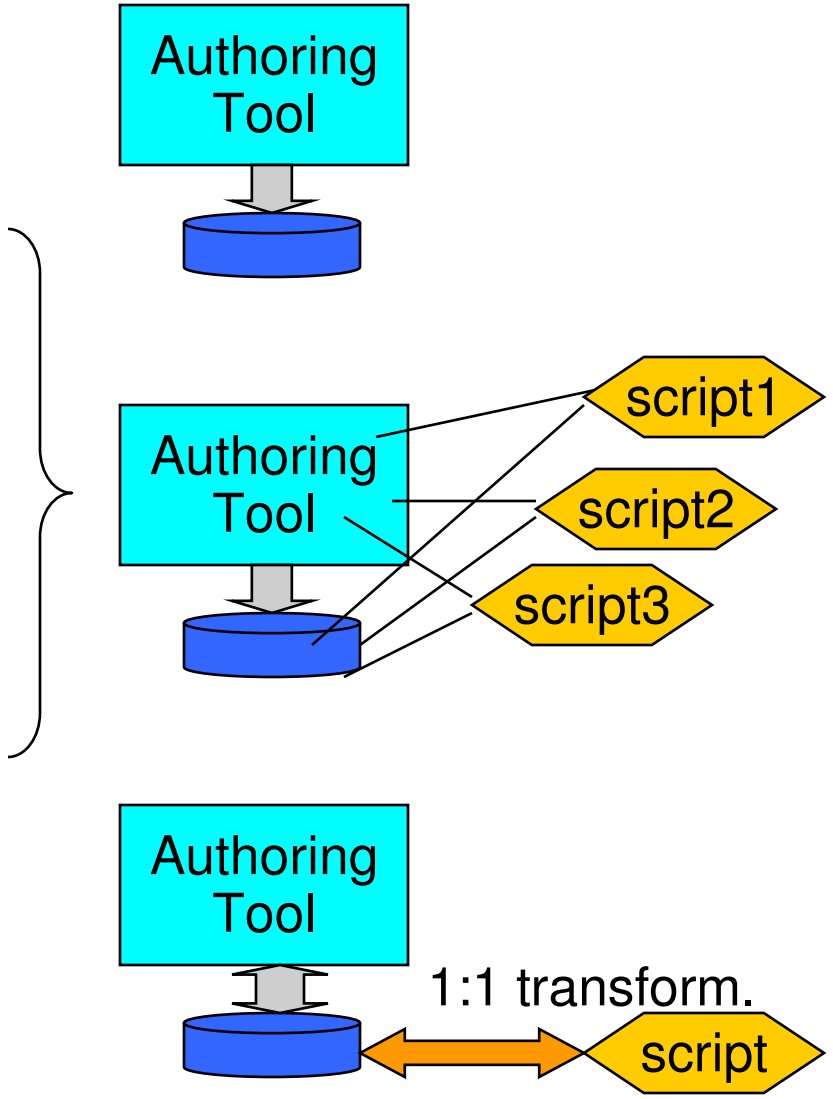
2 Development of multimedia applications

Scripting Languages for Authoring Tools

- *Script-less authoring:*
  Just an authoring tool
  Scripts/programming avoided
- *Local scripting:*
  Scripts added at various places
  in the authoring environment
  to enhance expressiveness;
  scripts are *context-dependent*
- *Global scripting:*
  Separate script files in addition to the
  file produced with the authoring tool;
  scripts are *self-contained*
- *Script-based development:*
  Authoring tool as a comfortable view
  onto a program (script);
  Whole application can be written as
  a script in a formal language

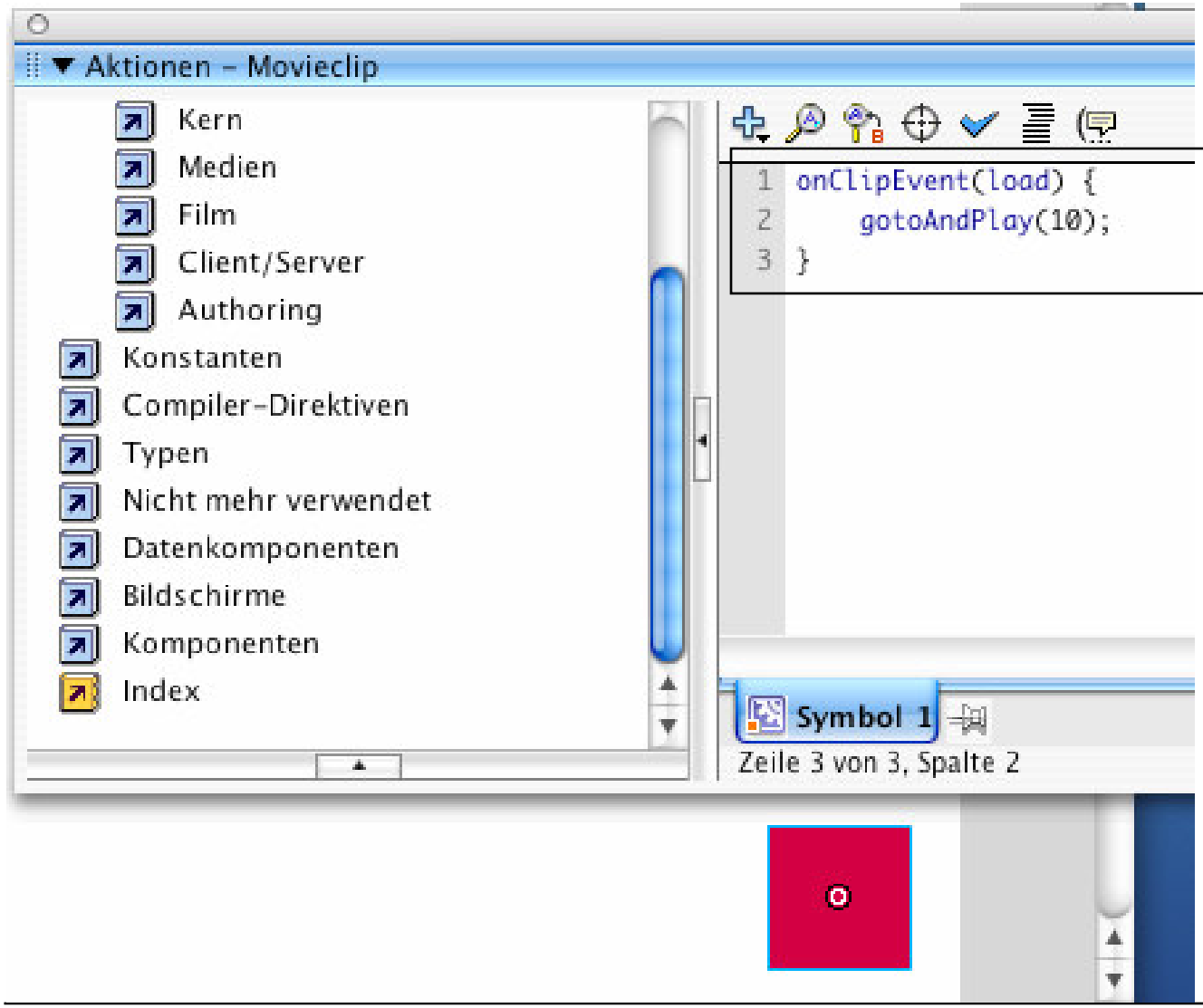


Flash Software Versions

- Flash 1 to 3:
  - Only very limited interaction
  - No scripting at all (script-less)
- Flash 4:
  - Beginnings of ActionScript (local scripting)
- Flash 5:
  - ActionScript 1.0 (local + simple global scripting)
  - Execution very slow
- Flash 6 = Flash MX:
  - Improved execution speed
  - Custom object classes (object-based programming)
  - Prototype objects, inheritance, no classes yet
- Flash 7 = Flash MX 2004
  - ActionScript 2.0
  - Global scripting
  - Java-like syntax, full class concept (object-oriented programming)

Script-based development for Flash: E.g. with KineticsFusion/RVML

Example of Local Scripting in Flash



Local event handler as script

Example of Global Scripting (ActionScript 2.0) Object-Oriented Programming

```
class Account {
  private var saldo:Number = 0;
  private var num:Number;
  public function Account(accnum:Number) {
      num = accnum;
  }
  public function debit(n:Number) {
      saldo -=n;
  }
  public function credit(n:Number) {
      saldo +=n;
  }
  public function getNumber():Number {
      return (num);
  }
  public function getSaldo():Number {
      return (saldo);
  }
}
```

ActionScript 1.0 and ActionScript 2.0

- ActionScript 1.0 (AS1)
  - Simple scripting language
  - Not built for large-scale programming
  - Implicit typing (inferred from variable name and value)
  - Object-based
- ActionScript 2.0 (AS2)
  - Only from Flash MX 2004 and Flash player 7 upwards!
  - Based on the ECMAScript standard (proposal 4)
  - Very similar to Java (Object-oriented)
  - Multiple classes, each defined in its own source file
  - Strict explicit typing
  - Case sensitive

> "ActionScript 2.0 can be called an object-oriented programming language, whereas previous versions were more modestly referred to as an object-*based* programming language, and that was only with the Flash MX version." William B. Sanders

Example of Global Scripting (ActionScript 1.0) Object-Based Programming

```
var Account = function(accnum) {
  this.saldo = 0;
  this.accNumber = accnum;
}

Account.prototype.debit = function(n) {
  this.saldo -= n;
}

Account.prototype.credit = function(n) {
  this.saldo += n;
}

Account.prototype.getNumber = function() {
  return (this.accNumber);
}

Account.prototype.getSaldo = function() {
  return (this.saldo);
}
```

Concepts of Object-Based Programming

- No classes:
  - Special objects *(prototypes)* serve as blueprints for newly created objects
  - No concept of a class!
  - Advantage: Dynamic changes to prototypes easily possible (e.g. adding a method - applies to all objects derived from prototype)
- ActionScript 1.0:
  - Variables can store everything, including functions and constructor functions
    - » This also applies to fields of objects
    - » No difference between attributes and methods
  - Constructor function as replacement for class
  - Constructor function associated with prototype (i.e. extensible by features)
- Flexibility vs. structuring:
  - Class-based, object-oriented languages (Java, AS 2) are easier to understand
  - Object-based languages (Smalltalk, AS 1) are more flexible and powerful
- Flash allows to some extent a mixture between the two styles!

2 Development of multimedia applications

File Types in Flash Development

- Flash Project (.flp)
  - Bundles the information required for a specific development project
  - Easily readable XML file
  - Mainly: Links to involved files
- Flash Movie (.fla)
  - Contains the main animation (timelines and symbols)
  - Unreadable binary file
  - Edited with the Flash authoring environment
- ActionScript (.as)
  - Contains an ActionScript class
  - Readable ActionScript ASCII file
  - Editable with any editor or with the built-in ActionScript editor of the Flash authoring environment
- Shockwave Flash (.swf)
  - Output format for Flash Player

Objects in Flash

- Everything is an object.
- *Visual objects:* Can be created and manipulated in the graphical authoring environment:
  - Objects of classes MovieClip, Button, TextField, Component, ...
  - Example: MovieClip object
    - » Member of the MovieClip class
    - » Has a TimeLine object where the classTimeLine defines methods like: **`play(), stop(), gotoFrame()`**
- *Non-visual objects:*
  - In particular objects of most developer-defined classes (“custom classes”)
  - Example: “Account” objects

- There is no conceptual difference between visual and custom objects !

How to Create an Object in Flash

- Visual Objects:
  - Visual Creation in the Flash authoring environment
  - Static, suitable for permanently existing objects (which may be invisible at times)
  - Creation of visual objects via method call
    - » Using specific methods like **`createEmptyMovieClip, duplicateMovieClip, attachMovie, ...`**

- Non-visual objects:
  - Explicit instantiation
    - » Script contains new-statement like in Java

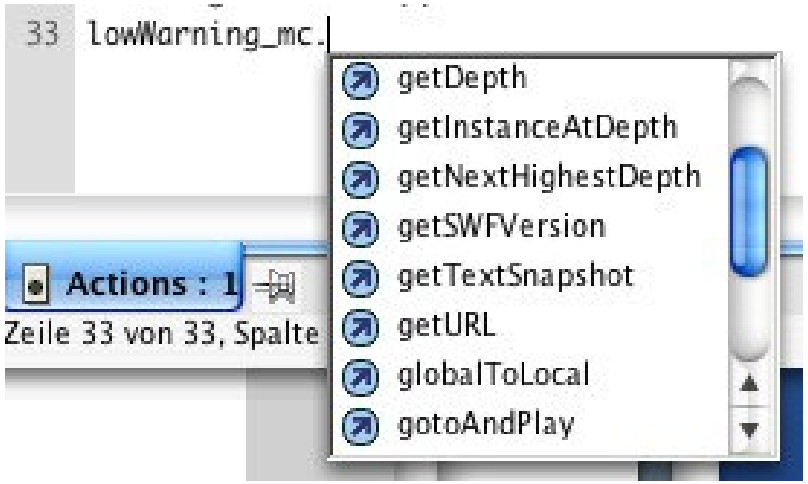
Strong vs. Weak Typing

- Weak Typing:
  - Variables and properties can be assigned different types of data at different times
  - Variables are declared without explicit type information
  - Example programming languages: BASIC, ActionScript 1.0
- Strong Typing:
  - Type information part of the variable declaration
  - All assigned values have to conform to the declared type at all time
  - Example programming languages: PASCAL, Java, ActionScript 2.0 (partially)
- Suffixing:
  - Only way in AS1 to get “code hinting”
  - See next slide

Type Hinting

- Naming convention for variables according to type of contained value
- Helpful mainly for weakly typed languages
  - "Hungarian notation" also used in C/C++, e.g. Microsoft standard
- Specific prefix or suffix of variable name indicates type
  - E.g. "variable names starting with 'p' indicate pointer values."
  - E.g. "variable names ending with '_mc' indicate MovieClip values"

- Information evaluated
  e.g. in programming environment
  - "Hinting" = interactive offer of adequate additions to currently edited programming text
  - For a variable named **`xy_mc`**, the special methods available for **`MovieClip`** objects are offered for selection



Types in ActionScript 2.0

- Types (= classes) for non-visual objects:
  - Array
  - Boolean
  - Number
  - Object
  - String
  - ...

  + custom classes defined by the developer using **`class { … }`**
- Types (= classes) for visual objects:
  - MovieClip
  - Button
  - TextField
  - Component

  For visual objects, type information by suffixing is recommended !
  (see below)

A Full List of ActionScript 2.0 Data Types

- Accordion
- Alert
- Array*
- Binding
- Boolean*
- Button**
- Camera**
- CheckBox
- Color*
- ComboBox
- ComponentMixing
- CustomActions
- DataField
- DataGrid
- DataHolder
- DataSet
- DataType
- Date*
- DateChooser
- Delta
- DeltaItem
- DeltaPacket
- Endpoint
- Error
- Function**
- Label
- LoadVars**
- LocalConnection**
- Log
- MediaController
- MediaDisplay
- MediaPlayback
- Menu
- MenuBar
- Microphone**
- MovieClip*
- MovieClipLoader
- NetConnection**
- NetStream**
- Number*
- Object*
- PendingCall
- PopUpManager
- PrintJob
- ProgressBar
- RadioButton
- RadioButtonGroup
- RDBMSResolver
- ScrollPane
- SharedObject**
- Slide
- SOAPCall
- Sound*
- String*
- TextArea
- TextField**
- TextFormat**
- TextInput
- TextSnapshot
- Tree
- TypedValue
- Video**
- Void
- WebService Connector
- Window
- XML*
- XMLConnector
- XMLNode*
- XMLSocket*
- XUpdateReceiver

* = already contained in Flash 5

** = added in Flash MX

Type-hinting suffixes in ActionScript 2.0

Array:	_array
Button:	_btn
Camera:	_cam
Color:	_color
Date:	_date
Error:	_err
LoadVars:	_lv
LocalConnection:	_lc
Microphone:	_mic
MovieClip:	_mc
NetConnection:	_nc
Sound:	_sound
String:	_str
TextField:	_txt
Video:	_video
XML:	_xml
XMLNode:	_xmlnode

Partial list !

Script Programming in Flash

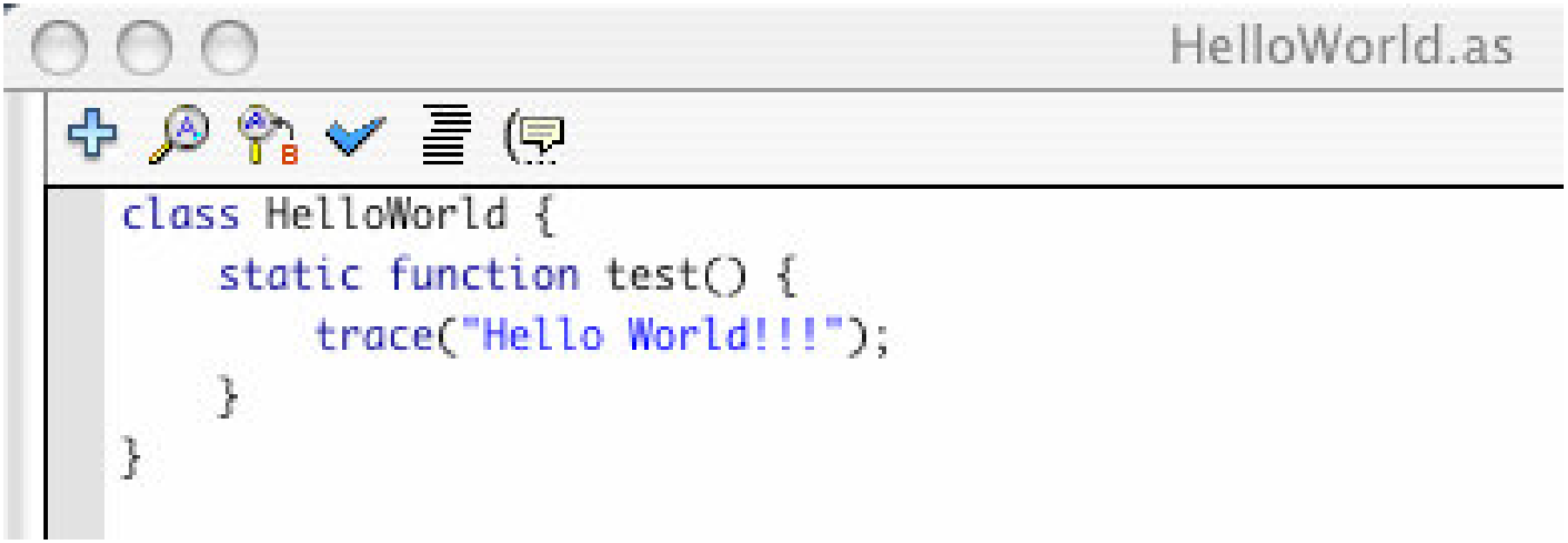
- Local scripting in Flash, static variant:
  - ActionScript code attached to certain frames of a certain timeline (see “Start Frame Code” pattern)
  - Code is executed as soon as the respective frame is displayed
- Local scripting in Flash, dynamic variant:
  - Action script code as event handler
  - Code always preceded with “on” or “onClipEvent” clause
  - Code is executed as soon as respective event takes place
  - (see section on interaction)
- Global scripting in Flash:
  - Separate “.as” files
  - Code is not executed at all if not bound into the application by some kind of local scripting!

Flash Pattern: Start Frame Code

- **Problem**: A Flash movie needs to carry out some ActionScript code which cannot be easily defined in a local, object-oriented style
  - Creation of objects on an application-global scale
  - Invokation of methods defined in external ".as" files
  - Assignment of methods to visible objects instantiated from the standard library (e.g. TextField)
- **Solution:**
  - Keep the "global code" in the main timeline (_root).
  - Add a separate layer (e.g. "code" or "actions") to the main timeline.
  - Add all "global" code to frame 1 of the newly created layer of the main timeline.
  - Advantage: There is just one place to inspect for the global code organisation.
- **Examples**:
  - Plenty found in literature

A HelloWorld Program in ActionScript

- ActionScript class in file “HelloWorld.as”

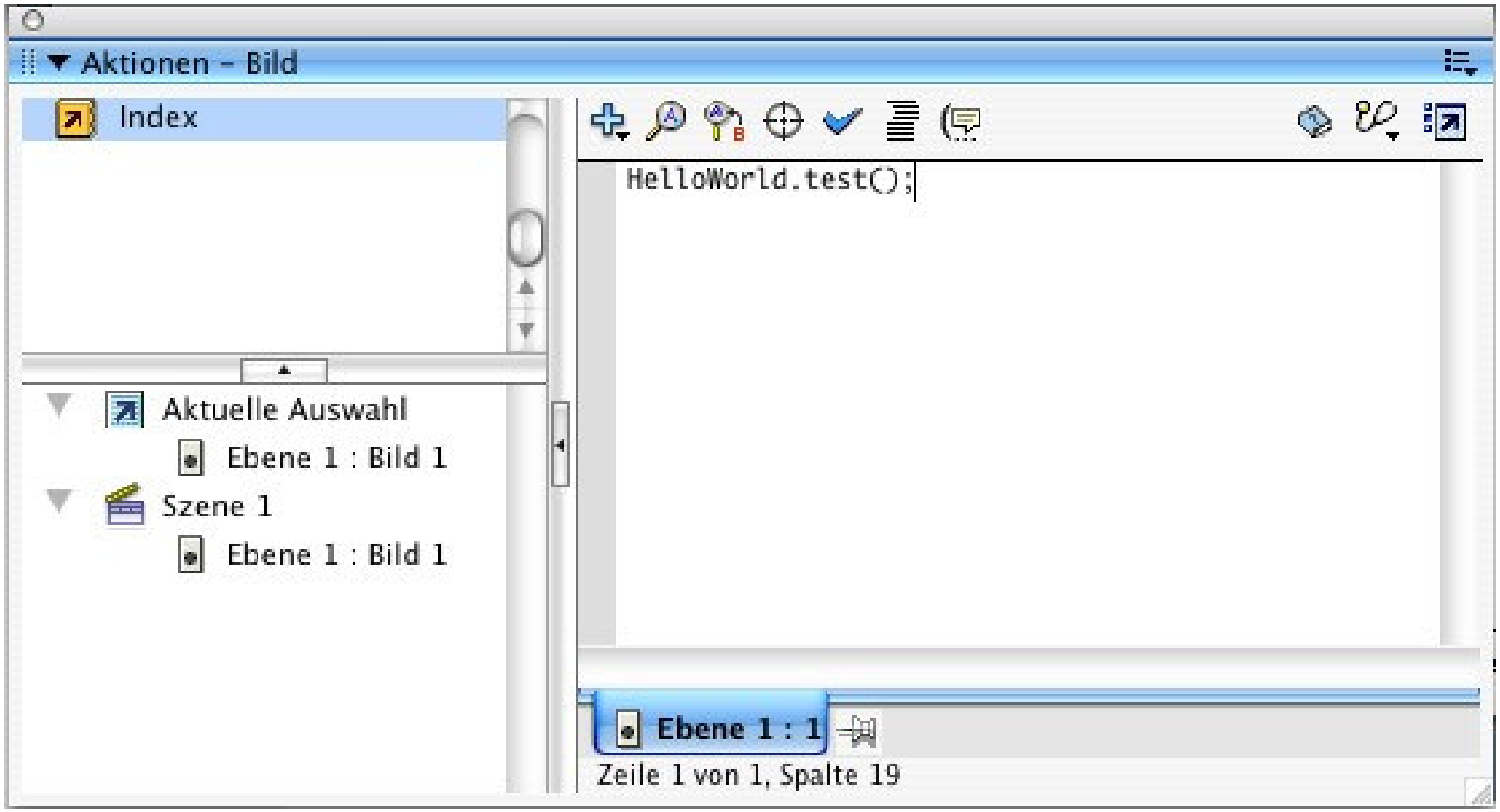


trace() Function

- `trace()`
  - Built-in function
  - Reports a message during runtime on the output console
  - Trace messages can be excluded from the exported SWF
    - » “File–>Publish Settings” / “Datei–>Einstellungen für Veröffentlichungen”

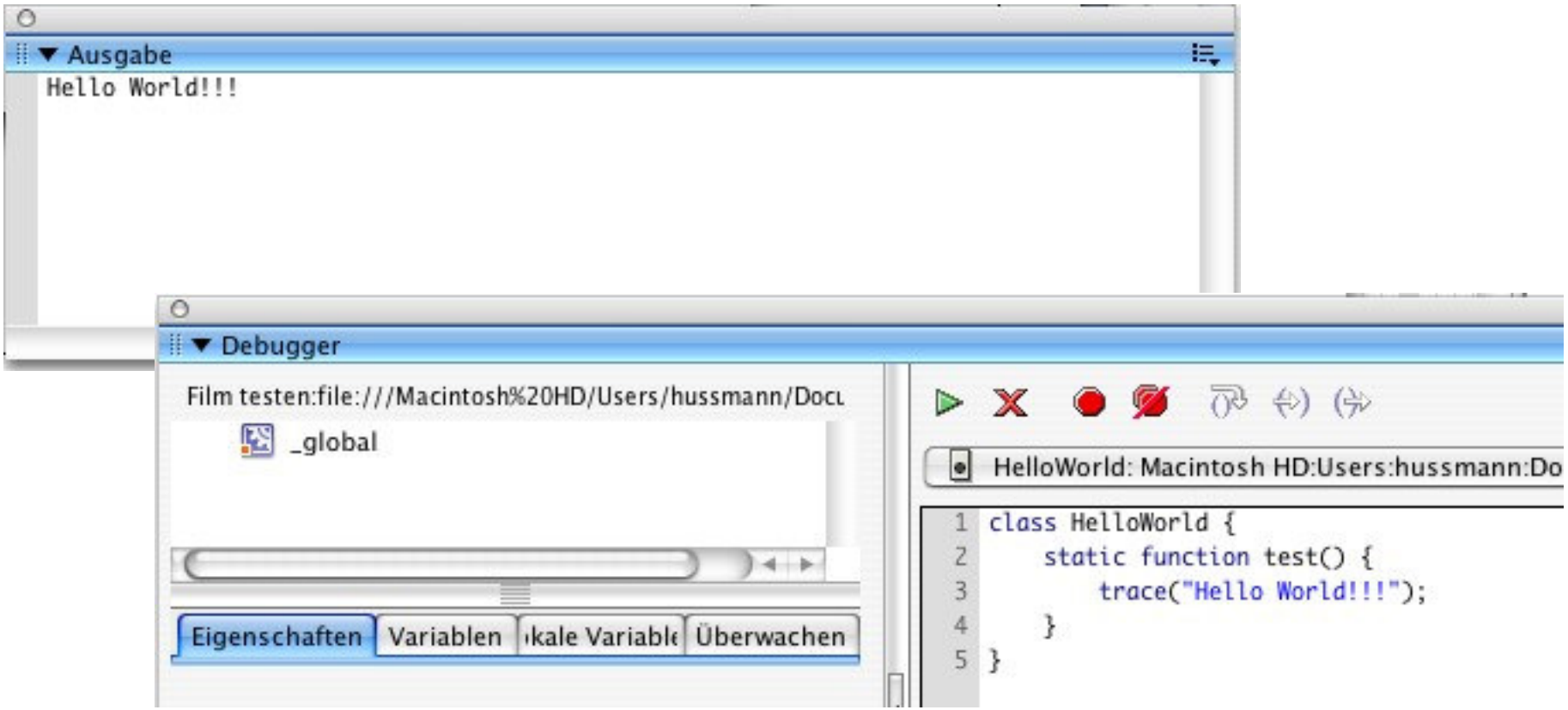
A Flash Movie Invoking the Hello World Program

- Flash movie “HelloWorld.fla”
  - Without any visible objects
  - ActionScript attached to Frame 1 of Scene 1



Running the Flash Hello World Movie

- Export as SWF file and start player
- Optional interactive debugger



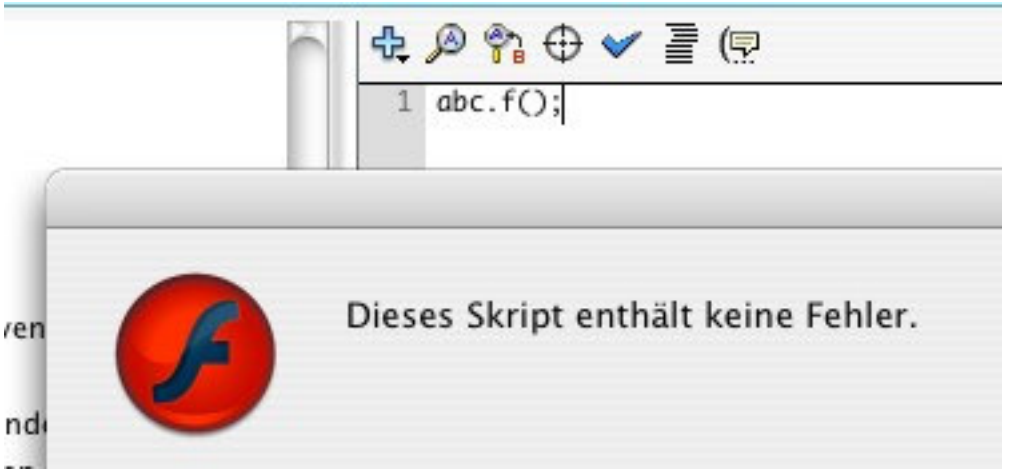
Functional Programming with ActionScript

```
class Fac {
  static function fac (n:Number):Number {
     if (n==0) {
          return (1);
     }
     else {
          return (n*fac(n-1));
     }
  }

  static function facCall (n:Number) {
     trace("fac("+n+")"+" = "+Fac.fac(n));
  }
}
```

Undefined Variables & Methods in ActionScript

- Not recognized as errors:
  - Referencing an undefined variable
  - Calling a method not defined in the class/type of a variable



- Purpose of “sloppy” definition/typing rules in scripting languages for authoring systems:
  - Product can be tested and presented even in incomplete state
  - Danger: Error detection by tool checks (eg type check) does not work properly any more

Modifying Attributes in ActionScript

- All visible objects come with a predefined (more or less large) set of attributes
  - Example: “_x” and “_y” for screen position
- ActionScript code can move visible objects around the screen by modifying these attributes
- Example:
  - Modifying an object (with an independent timeline)
  - In Frame 1 (key frame) : `inst_mc._x = 10; inst_mc._y = 10;`
  - In Frame 6 (key frame): `inst_mc._x = 20; inst_mc._y = 20;`
  - In Frame 11 (key frame): `inst_mc._x = 40; inst_mc._y = 40;`

Example RVML: Nested Timelines, ActionScript

```
...
<Definitions>
  <MorphShape id='inst_mc.MorphShape_1'> ...
  </MorphShape>
  <MovieClip id='inst_mc'>
    <Timeline frameCount='5'>
      <Frame frameNo='1'>
        <Place name='inst_mc.MorphShape_1' depth='1' />
       </Frame>
    ...</Timeline>
  </MovieClip>
</Definitions>
<Timeline frameCount='11'>
  <Frame frameNo='1'>
    <Place name='inst_mc' depth='1' instanceName='inst_mc'>
      <Transform translateX='199.0' translateY='98.0' />
    </Place>
    <FrameActions><![CDATA[
inst_mc._x = 10;
inst_mc._y = 10;
]]></FrameActions>
  </Frame>
...
```