The Macao Tutorial

Version 1.2

Peter Trauberg

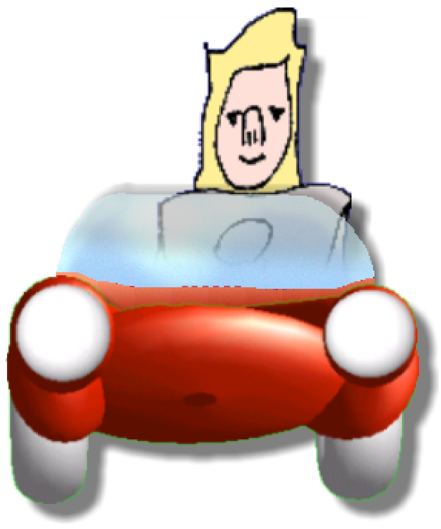


Table of Contents

Index of Figures

Index of Tables

1 Introduction

1.1 The Vision of Macao

Macao is build to help you tell interactive animated stories with a minimum of effort. You can make the stories available on almost every PC without installation. Put your Macao story or animated web page as a static web site to the Internet. Every user can explore it with most state-of-the-art browsers. No plugin is needed, because Macao works with dynamic HTML.

Using Macao, you can make a visitor explore your homepage rather than just view it. The visitor can steer a character or drive a car through your page. These objects can be used to communicate with other characters or objects. You can give the visitor the opportunities to collect items, use them on other items and discover hidden doors or treasures. Your Macao site may consist of a single page or of a large site of linked pages, which work together. You can animate your web shop, let tell characters about your hobbies or you can tell an epic story. You can build in puzzles or just build a landscape-like diorama to walk through.

As Macao consists of HTML, you can use your HTML-skills to design your page. Use your digital camera to take pictures, convert them to transparent GIFs or PNGs and put them as Macao objects to your page. With a few commands you can define paths at your page, which an object can use to walk along. You can use the Road Editor to create roadmaps for cars and other vehicles. By adding Talk-Items to objects you can define conversations and make the characters act on objects.

This tutorial shows you the construction of two basic types of pages. You will see how few functions it needs to build rich interactive applications.

Macao consists of an object oriented API written in JavaScript. The API is documented in Java-Doc-style. The number of classes in the API is kept small, to make the programming clear and simple. The distribution contains a few controllers, which you can put at your page. The controllers are serving as inventory for collected items or helping the user to "talk" to the characters.

Currently Macao comes with a set of images for two characters and other images with objects, which you can use at your own pages. You can build new characters by only a small number of images (at least one) or you can create characters with many animated features. A little tool helps you to rotate images to animate vehicles.

Macao is Open Source under the terms of the GNU General Public License. So you can use it for free on your homepage or in any kind of distribution.

1.2 License

1.2.1 In Addition

In Addition you may use all artwork and characters, which come with the distribution, in your own application. You may use it, change it and further develop it.

1.3 Requirements

1.3.1 Skills

To develop websites with the Macao API you need knowledge in JavaScript or an object oriented programming language. You also should have know-how in the programming languages HTML/DHTML and CSS (cascading style sheets). You can find tutorials for these techniques at http://www.w3schools.com. For language references see the chapter Resources.

1.3.2 Documentation

The documentation for Macao consists of this tutorial and the **Full-** and the **Designer-**Version of the **API Documentation**.

Because in JavaScript has no technique to protect private class members against access, the documentation has been split in two parts. The Designer-Version documents the classes and methods, which you need to build a Macao application. The Full-Version also documents the internal used fields and methods. You may consult the full documentation, if you want to modify or extend the core Macao functionality.

So it's recommended to start with the Designer-Version of the API Documentation. Consult the documentation to get detailed information about the classes and methods. Every method has a description about where it is supposed to use.

1.3.3 System Requirements

The Macao API supports the following browsers:

- Internet Explorer 5.5 and above
- Mozilla 1.3 and above
- Firefox 0.9 and above
- Netscape 7 and above
- Opera 7 and above

Currently the **Konqueror** is not supported, because the animation is to heavy for this rendering engine.

So far the Macao API is developed and tested with Mozilla 1.4 to 1.8, Firefox 1.0 to 1.5, Internet Explorer 6 and Opera 7 on Windows 98/NT/XP and with Mozilla 1.7 on Linux.

2 Download and Installation

Download the Macao SDK zip-file. Extract the zip-file to your local file system.

Open the file index.html of the Macao-directory with your browser. This opens the local files of the homepage. Via the homepage you can browse the documentation or start the sample application Discover Macao.

2.1 The Directory Structure

The Macao directory has the following sub-directories:

- **core**: This directory is containing the core Java Script libraries of Macao.
- **optional**: This directory is containing some stuff, which you may use in your application.

  When you deploy your Macao based web pages to a web server, you should only upload some of the subdirectories with the content, which you have used in your application. The subdirectories currently are:

  - **cards**: This subdirectory contains the resources and definitions for a standard set of cards.
  - **cars**: This subdirectory contains images for cars.
  - **characters**: This subdirectory contains images for characters.
  - **controllers**: This subdirectory contains HTML-pages with controllers, which you can include in your frameset.
  - **landscape**: This subdirectory contains images, which can be used to build landscape pages.
  - **resourceExport**: This subdirectory contains the resource export tool, which can be used to internationalize the application.
  - **road**: This subdirectory contains a standard set of road types.
  - **roadEditor**: This subdirectory contains the files of the Road Editor.
  - **room**: This subdirectory contains images, which can be used to build pages showing rooms.

- **examples**: This directory contains subdirectories with examples. The examples are discussed in this tutorial or in the API-documentation.
    - **tutorial**: This subdirectory contains the example, which is discussed in this tutorial.
- **tools**: This directory contains subdirectories with offline tools.
- **doc**: This directory contains this tutorial. And it contains subdirectories with the Designer- and the Full-API documentation of Macao.
- **discover**: This directory contains the demo application Discover Macao.
- **homepage**: This directory contains the files of the Macao homepage.

3 The Packages and Classes

This chapter gives you an introduction into the JavaScript packages of Macao. It tells you, which of the packages are used in which kind of Macao HTML pages. Each package defines some classes, which are documented in the **API Documentation**. Later you will learn step by step how to use these classes.

Note Compressed versions of the core packages are available in the directory core/compressed. In the compressed files the white space and the line comments are removed. This saves about 30% of the size. You can create compressed versions by your own using the tool **Compress** (tools/compress/compress.hta). Currently this tool works only on MS Windows.

3.1 The Package Kernel

The package Kernel is implemented in the file **core/kernel.js**. It contains the class **MacaoObject**. This class is the base class for every object, which is placed and moved on a Macao web page. It also contains the helper classes **MacaoLook**, **MacaoEvent** and **MacaoBubble**. The Kernel encapsulates most of the browser dependent functionality.

Include the package Kernel in every page except pages, which are only containing frameset definitions.

3.2 The Package Dynamic

The package Dynamic is implemented in the file **core/dynamic.js**. Load this package in a page, to extend the class **MacaoObject** with dynamic functionality like walking and talking.

The package contains the class **MacaoTalkItem**, which is used to define conversations and other actions of objects.

The classes **MacaoNet**, **MacaoNode** and **MacaoConnection** of the package are used to define networks of paths, at which objects can walk. Walking means walking of characters as well as driving of cars and other vehicles.

The package Dynamic is needed in every main content page. It is not needed in controllers- and inventory pages. Load this package after the Kernel package.

3.3 The Package Road

The package **Road** is implemented in the file **core/road.js**. This package is needed in every main content page, which has to display a road map. It is not needed in controllers- and inventory pages. Load this package after the dynamic.js package.

Add a road to your page, to create a net, which can be used by cars and other vehicles. A road is just an easy way to create a network with a lot of nodes and connections.

The class **MacaoRoadGrid** of this package is used to make the road visible at the page.

3.4 The Package Menu

The package Menu is implemented in the file **core/menu.js**. The package is needed in every page that displays controllers like a **TalkController**, a **SentenceController** or the **Inventory**. Normally controllers are not included in a content page but in extra frames. Include this package after the package kernel.js in a page.

The classes **MacaoMenu** and **MacaoMenuItem** are used to display buttons on the page, which can contain images and/or text. The other classes in this package are controllers, which are inherited from MacaoMenu. The user uses controllers to interact with the MacaoObjects.

3.5 The Package StorageManager

The **StorageManager** is used to store values to transfer them between the content pages. It is implemented in the file **core/storageManager.js**. Include the package in the top most frame of your frameset.

If your application uses no frames, include the storage manager in each page. In this case, the storage manager can't transfer values from one page to another page.

3.6 The Package Frameset

The package Frameset is to be included in pages, which only consist of a frameset. For this pages this is the only package, which is to be included. The package registers the frameset page to the event system. The package is implemented in the file **core/frameset.js**.

3.7 The Package Persistence

The package Persistence implements the functionality for saving and loading the game score. The package is implemented in the file **core/persistence.js**. When used this package is to be included in the page with the StorageManager.

3.8 The Package Cards

The package Cards can be used to create a card game. The package is implemented in the file **core/cards.js**.

4 Creating a Macao Application

In this chapter we will create the basics of a Macao application. This consists of the page index.html, which contains the frameset, and one content page.

You can find the source code of this example in the directory examples/helloWorld.

4.1 Creating your Application Directory

Much functionality in Macao is related to the **Macao base directory**. This is the directory with the subdirectories "core", "optional" and others. You should place your own application in a subdirectory of the Macao base directory. You may also create subdirectories for each page in your application directory.

Other examples are located in subdirectories of the directory **examples**. So they have to use the path "../../" to reference the Macao base directory. In these examples all files of an example are located in one directory.

4.2 Creating the Frameset

Normally your Macao application needs a frameset. The frameset is used to keep data in the Storage Manager to pass them from page to page. The frameset is also used to display controllers in their own frame. Controllers are sets of controls, which are used by the user to interact with the objects at the content page. While the user scrolls the content page, the controllers stay visible.

We are creating a page index.html, which contains the frameset. The frameset has three frames with the names **content**, **controller** and **inventory**.

The frame **content** will contain the content pages. The content pages will show a landscape with a road map or rooms of any type.

In the frame **controller** we include the page optional/controllers/controllers.html. This page contains the **SentenceController**, the **TalkController** and the **SteeringController**. So the user can use these controllers to interact with the objects at the content page. See the API Documentation of the package menu on how to customize controllers or implement your own controllers.

In the frame **inventory** we include the page optional/controllers/inventory.html. This page contains an inventory object. We can use it to add or remove items to the inventory. The user can use the items in the inventory to build commands in the SentenceController.

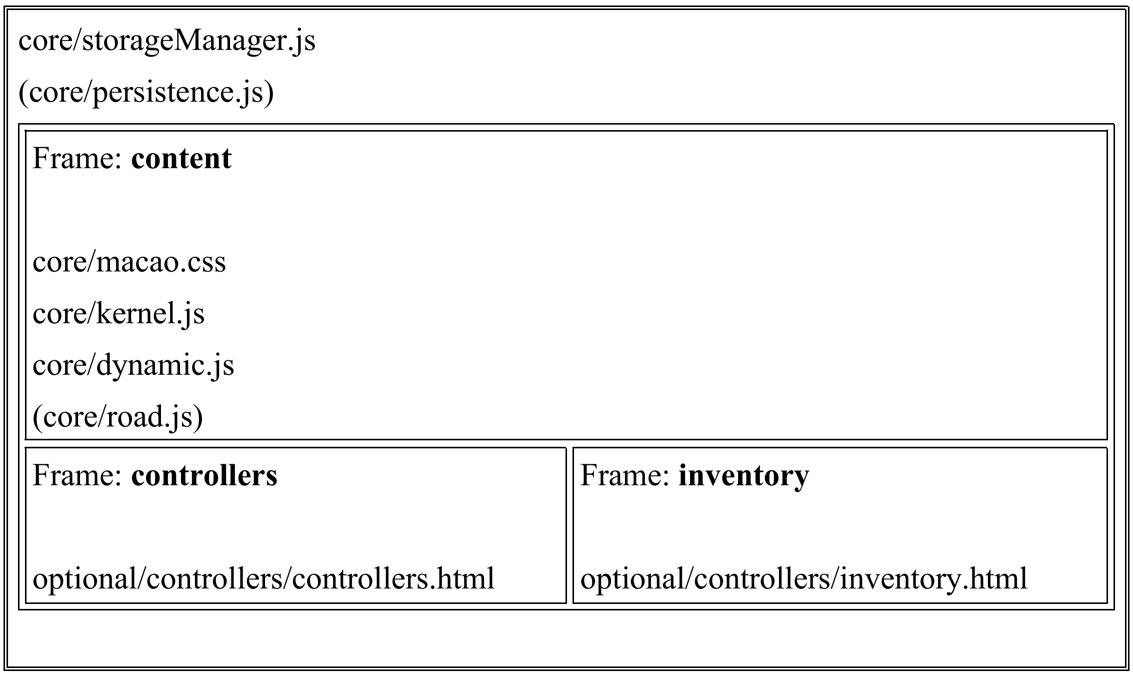


Figure 1: The Standard Frameset

Here is the source of the frameset page:

```
<html>
<head>
<meta http-equiv="expires" content="43200">
<title>My Application</title>
<script language="JavaScript" src="../../core/storageManager.js"
type="text/javascript"></script>
</head>
<frameset
        rows="*,100"
        border=0 frameborder=0 framespacing=0
>
        <frame
                name="content"
                src="page1.html"
                marginheight=0 marginwidth=0
        >
        <frameset cols="50%,45%">
                <frame
                        name="controllers"
                        src="../../optional/controllers/controllers.html"
                        marginheight=0 marginwidth=0 scrolling=no
                >
                <frame
                        name="inventory"
                        src="../../optional/controllers/inventory.html"
                        marginheight=0 marginwidth=0
                >
        </frameset>
</frameset>
</html>
```

The page with the topmost frame, which is the page index.html, has to contain the **StorageManager**. The StorageManager is used to keep data while the content pages and controller pages are changed. So the pages can communicate through the StorageManager. When the page with the StorageManager is closed, the data are lost.

4.2.1 Omitting Framesets

You can build Macao pages or applications without framesets. But then the only way to pass information between the pages is using parameters in the URL. Because there is no place for the controllers here, you have to design the page in a matter, that it can be controlled only by mouse clicks to the page.

When you create a page without using a frameset, you need to include the package **StorageManager** in the page. Else the use of the StorageManager functions would raise an error.

4.3 Creating a Content Page

In a content page you have to include the standard packages and a standard CSS-style-sheet. You need to set an expiry-duration and the base path. At least we will add an object to the page.

4.3.1 Setting an Expiry Duration

It's recommended to set an **expiry-duration** in every page. An expiry-duration of 12 hours will force the reload of the page files every day, at which the user surfs to your page. Setting an expiry-date speeds up the Macao application and reduces the traffic at your site, because the data are not to be loaded for every page again and again.

Note The expiry-duration is only a hint to the browser. The user can change this behavior by changing the browser settings. Watch the traffic of your website, when you publish your application to the Internet, because a Macao application can contain a lot of multimedia stuff for download.

Note If you set no expiry duration, some early versions of the Internet Explorer tend to ever use the very first version of the page.

Use the following HTML to set the expiry duration to 12 hours.

```
<meta http-equiv="expires" content="43200">
```

4.3.2 Setting the Base Path

In the head of the page you have to set the base path. The base path is a relative path between the location of the page file and the Macao base directory. For example, if you have placed your page in the subfolder myapp, set the base path to “../”.

The Macao packages need the base path to calculate the location of other Macao files.

Note The HTML <base> tag can't be used for this purpose, because it doesn't support the parent directory “../”.

Assign the base path to the global variable **basePath**. Set the base path, before you load the packages to the page. Use the following HTML to set the base path. Adjust the path according to the location of your page:

```
<script>
var basePath = "../../"
</script>
```

4.3.3 Including the Standard Packages

A content page has to include the packages **core/kernel.js** and **core/dynamic.js** or their compressed versions. The package **Kernel** defines the basic class **MacaoObject** and creates an invisible **bubble** object, which is used to display the text which an object is saying. The package **Dynamic** adds functionality for walking and talking to the class MacaoObject.

Include also the standard CSS style sheet **core/ macao.css**. It contains the CSS style definition for the bubble and for the road grid, which we will use later to create a road map.

Use the following HTML to include the packages. Adjust the paths of the packages according to the location of your content page. The base path, which you set before, does not affect these paths, because they are entered directly into an HTML element tag.

```
<link rel=stylesheet type="text/css"
href="../../core/macao.css">
<script language="JavaScript"
src="../../core/kernel.js" type="text/javascript"></script>
<script language="JavaScript"
src="../../core/dynamic.js" type="text/javascript"></script>
```

4.3.4 Adding an Object

To add an object to the page, we create an object of the class MacaoObject. The object gets the internal name “Hero” and the title “Arnold”.

Then we assign an image to the object. The URL of the image has to be relative to the base path, which was set before. Later, when an object has to change its image dynamically, we will prefer looks to set images to an object.

We activate the ability of the object to walk to the position, where the user clicks with the mouse.

At last we send an event to the image to let it say “Hello world!” When the event is delivered to the object, it calls the method, whose name fits to the provided event type “say”. Even if we could call the method of the variable hero directly, it is best practice to send an event. The variable "hero" will go out of scope, but an event can reach the objects in all frames by their names. Later we will use MacaoTalkItems to define the text an object is saying.

```
// creat a new object
var hero = new MacaoObject("Hero", "Arnold", 200, 200)

// set an image to the object
```

```
hero.setImage("examples/helloWorld/Arnold.gif", 43, 41)

// activate the ability to walk to the mouse click position
hero.setApproachClick(true)

// send an event to the hero object
sendEvent("say", ["Hello World!"], "Hero")
```

5 Some Tips for Using JavaScript

This chapter gives a few tips for using JavaScript, which you need to understand this tutorial.

5.1 Arrays

In JavaScript you can create an array just by a pair of square braces. This code creates an array with two elements.

```
var myArray = ["Hello World!", true]
```

5.2 Optional Parameters

JavaScript doesn't check if you provide the appropriate number of parameters, when you call a method. When you omit trailing parameters, then this is equal to setting them to null or false. In the API Documentation the optional trailing parameters are marked optional.

The examples in this tutorial often don't mention the optional parameters. So refer to the API Documentation to get the full method signature with all optional parameters.

5.3 The Use of Semicolon

You don't need to terminate each JavaScript command by a **semicolon**. Just enter a line break. None of the supported browsers needs semicolons.

You also don't need to surround your JavaScript code with an **HTML comment** <!-- --> when it is entered in an HTML page. All browsers, which are supported, know JavaScript and will not try to display the source code as visible text.

5.4 Using the JavaScript Console

In the Mozilla Browsers you can use the menu *Tools – Web Development – JavaScript Console* to open the **JavaScript Console** . The console lists errors and can help you to debug your application.

The "old" Mozilla Browser has also the JavaScript debugger **Venkman**. You can use the debugger to debug your application.

6 Creating a Page with a Road Map

In this chapter we will create a page with a **road map** and add a car to it. The car will drive to the position, to which the user clicks with the mouse. You will learn how to use the **Road Editor** to add and remove road elements. Then we will add a building and assign the building to a road element. When the user drives the car to the road element, another page will be opened, which by example may show the building.

Note If you want to create a card game, you can proceed with chapter 11. Maybe you will come back later to learn more about Macao.

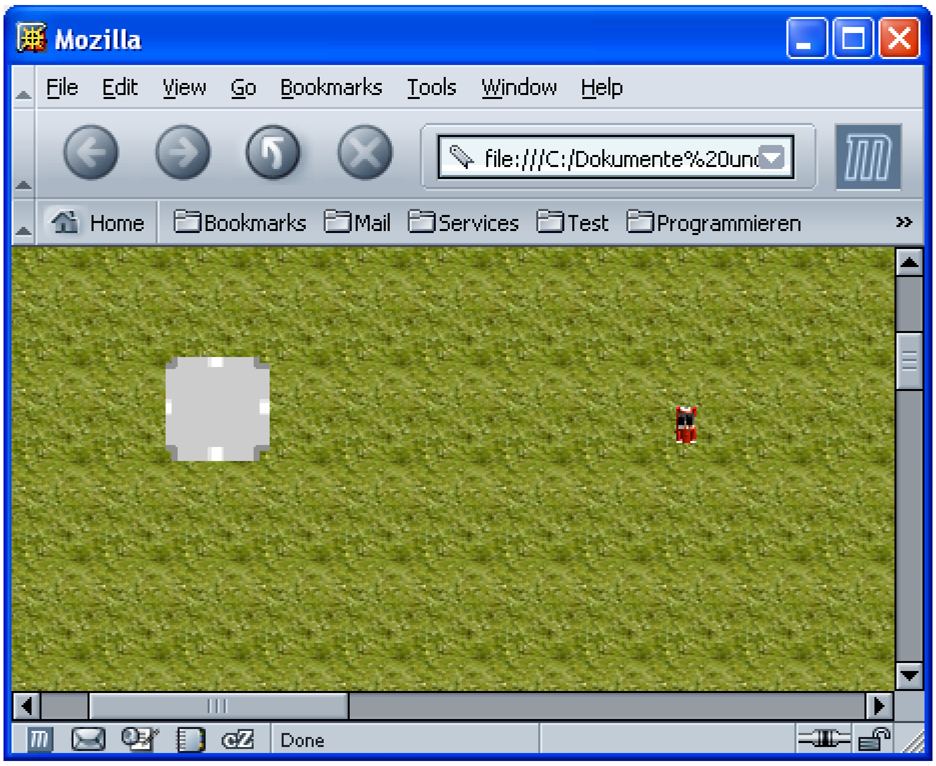

Mozilla
File Edit View Go Bookmarks Tools Window Help
file:///C:/Dokumente%20und
Home Bookmarks Mail Services Test Programmieren
Done

Figure 2: A simple road page

You can find the source code of this example in the directory examples/road. Figure 2 shows the simple road page, before it is edited using the Road Editor.

Note Because this example needs no controllers, the frameset is omitted. So the package core/storageManager.js has to be included in the content page. If you use the example in a frameset, remove the Storage Manager from the page.

6.1 Creating a Page with a Road Grid

To add a road to the page, we need to include the packages **core/kernel.js**, **core/dynamic.js** and **core/road.js** to the page. We also need to include a package, which defines the road types. Macao comes with the package **optional/road/roadTypes.js**, which defines a number of standard road elements.

To create the road, we first have to call the function **MacaoPage.setRoadMetrics()**, to define the position of the road grid and the size of the road cells. By defining the size of the road cells, we can zoom the display of the road.

Then we have to add road elements to the road. The elements are added using the method **MacaoPage.createRoad()**. When calling this method, you provide the type of the element, it's position in grid units, it's orientation and optionally it's name. Because it's toilsome to create all the method calls by hand, you can use the Road Editor (see below). Use the Road Editor to add the road elements with the mouse. Because the Road Editor is web based, it can't write its result back to your source file. So you have to export the result of your design work with the export function of the Road Editor. Then you have to paste it via the clipboard into the code of the page. Because we want to use the Road Editor, we initially add only one road element to our new road. Replace this later by the export of the Road Editor.

After all road elements have been added, we have to link them by calling the method **MacaoPage. linkRoads()**. When you add a road element, it creates its nodes and creates its internal connections. Linking the roads connects adjacent road elements, so a car can 'walk' from one road element to another.

At last we have to create the **MacaoRoadGrid**. The road grid is a MacaoObject, which makes the road visible, displaying its graphics using an HTML table. The position of the road grid is already defined by the call of the method setRoadMethrics(). You can only add one road grid per page.

The following source code creates a page with a road, which has a single road element. Use the road editor, to add more road elements.

```
<head>
<meta http-equiv="expires" content="10000">
<script>
var basePath = "../../"
</script>
<link rel=stylesheet type="text/css" href="../../core/macao.css">
</head>
<body background="../../optional/images/grasTile.gif">

<!-- add the required packages -->
<script language="JavaScript" src="../../core/kernel.js"
type="text/javascript"></script>
<script language="JavaScript" src="../../core/dynamic.js"
type="text/javascript"></script>
<script language="JavaScript" src="../../core/road.js"
type="text/javascript"></script>
<script language="JavaScript" src="../../optional/road/roadTypes.js"
type="text/javascript"></script>
```

```
<script language="JavaScript">

// set the road metrics
setRoadMetrics(20, 200, 26)

// add a single road element
// replace the following line with the Road Editors export
createRoad("RoadDJunc", 5, 1, 0,  null)

// link the road elements together
linkRoads()

// create a road grid to make the road visible
var roadGrid = new MacaoRoadGrid(50, 40)

</script>
</body>
```

6.2 Adding Cars

To add cars to the page, we create **MacaoObjects**. We have to call a few methods to make the object behave like a car. First we add a car, which is controlled by the user. Then we add a number of cars, which are randomly “walk” through the road net.

There is one common functionality, which is used to make cars drive and characters walk. This is called '**walking**'.

When creating the object, we define the cars internal name, its title and position at the page. The title is displayed as **tool tip**, when the user moves the mouse over the object. The name must be unique at the page. It is used to send events to the object.

Note You can call the method **setToolTip(null)**, to remove the tool tip of an object.

We call the method **MacaoObject.bindToNet()** to assign the car to the road net. The net, which is build by the road elements, has the name “**Road**”.

We call the method **MacaoObject.setApproachClick()** to make the car “walk” to the position, where the user clicks with the mouse.

We call the method **MacaoObject.setWalkCharacteristics()** to change the objects maximum velocity and acceleration. The values are set in pixels per walk step. A step by default has the duration of 150 ms plus the time the browser needs to calculate and display the object's next position. You can change the duration by calling **MacaoPage.setWalkInterval(),** but the velocity and acceleration are recalculated internally. The maximum velocity is set to 7 pixels per step. The acceleration is set to one pixel per step.

We call the method **MacaoObject.setScrollVisibleOptions()** to keep the car visible while it walks. If the car walks out of the visible client area, the page is scrolled automatically to make the car visible again.

We call the method **MacaoObject.setCollisionBreak()** to make the car stop, when it approaches another car from behind. The other car needs also to have this functionality activated. This only works, if the car approaches from behind. Else if two cars were approaching frontal, this functionality would cause a deadlock.

The call of the method **MacaoObject.addController()** assigns the **SteeringController** to the object. If the SteeringController is loaded into another frame, the user can use it to control the car in addition to use the mouse clicks at the page. So the use of the Steering Controller is optional.

Because DHTML is not able to rotate images, a car object has to show a different image for each direction. Each of the images is defined by a **MacaoLook** object. Look objects can be added to the MacaoObject by calling the method **MacaoObject.createLook()**. To create and add the looks for all directions with one call, we use the method **MacaoObject.addLookBunch()** instead. We take the images for the directions from the subdirectories of **optional/cars**.

The **bunch type** defines the purpose of the images. There are the bunch types "**Stand**", "**Say**" and "**Walk**" available. The object automatically searches the looks of the bunch types to get a fitting look and shows its image. Because the car has to look the same when walking and standing, we only need to define the looks for the bunch type Stand. These looks will be also used for walking.

The bunch **depth** (5) defines, how many images are loaded to display the directions. A depth of 5 means 2 to the power of 5, which are 32 images. These images have to be available at the defined location. See below how to easily create those images from a master image.

```
// create the car
var car = new MacaoObject("Car", "My Car", 500, 500)

// let the car use the road net
car.bindToNet("Road")

// make the car drive to the user's mouse click
car.setApproachClick(true)

// set maximum velocity and acceleration
car.setWalkCharacteristics(7, 1)

// make the page scroll, when the car leaves the visible area
car.setScrollVisibleOptions(true, 20)

// make the car stop when it approaches another car from behind
car.setCollisionBreak(true)

// link the car to the steering controller
car.addController("SteeringController")

// add the images for the directions
car.addLookBunch(
     car.BUNCH_TYPE_STAND,
     "optional/cars/red/carRed", 5, ".gif", 20, 20
)
```

Now lets add a number of cars, which are walking randomly through the road net.

We call the method **MacaoObject.setWanderAround()** to make the cars walk through the net, to which they are bound. We define a base duration of 5000 ms to wait, when a target node is reached. After this time the car will choose another node of the net to walk to. You can also provide an array with nodes of the net, which the cars have to use as targets of their random walk.

```
var carWander
var carNo
```

```
// create five cars
for(carNo = 1; carNo <= 5; carNo++) {

      // create and position a car
      carWander = new MacaoObject(
            "CarBlue" + carNo,
            "Wandering Car " + carNo,
            400,
            200 + (30 * carNo)
      )

      // add the blue car images
      carWander.addLookBunch(
            carWander.BUNCH_TYPE_STAND,
            "optional/cars/blue/carBlue", 5, ".gif", 20, 20
      )

      // bind the car to the net
      carWander.bindToNet("Road")

      // set maximum velocity and acceleration
      carWander.setWalkCharacteristics(7, 1)

      // activate collision break
      carWander.setCollisionBreak(true)

      // make the car wander around
      carWander.wanderAround(5000)
}
```

6.3 Defining a Look Bunch

Lets take a closer look at **bunch types**. A bunch uses a special naming to name images and their MacaoLook objects which are used to show an object standing, walking or talking in several directions. The filename of such an image has the following form:

baseFileName bunchType directionName [phaseName] extension

The name of a look has the form

bunchType directionName [phaseName] extension

An example for a file name is:

carBlueStand01111.gif

The **baseFileName** is the part of the filename, which you can freely choose. You have to provide this part with the parameter imageBase of the method addBunchType(). We had used this method before to add all the images of a bunch to an object. The **bunchType** has to be one of the values “Stand”, “Say” or “Walk”. **“Stand”** is the basic type. These are the pictures, which are used to display the object, while it is standing. **“Say”** is the bunch type, which images will be used, when the object is talking. If you omit the bunch type Say, the images for standing are displayed instead. Use the bunch type “**Walk**” to add images, which are used to display the object, while it is walking. If you omit this images, the images from the bunch type Say or Stand are used for walking. Because a car looks almost the same, when it stands or moves, we added only the images for the bunch type Stand.

The **directionName** is a special part, consisting of "0" and "1" characters. These characters are used to code the direction of the object on the image. Figure 3 shows the code for up to 32 directions. When you add a bunch to an object, you don't have to provide all the image names, but you only have to provide the **depth**. Where the number of images is 2 to the power of depth. So one call of the method addBunchType() will calculate the names of all the images of the bunch and load them as MacaoLook objects to the object. Another advantage of this coding is, that you only need to reduce the depth parameter in order to reduce the number of images to be used. The images don't have to be renamed. For cars and other objects, which are to be shown in birds view, there is the tool **rotate**, which you can use to create all the images of a bunch from a master image. This tool is described in the next chapter.

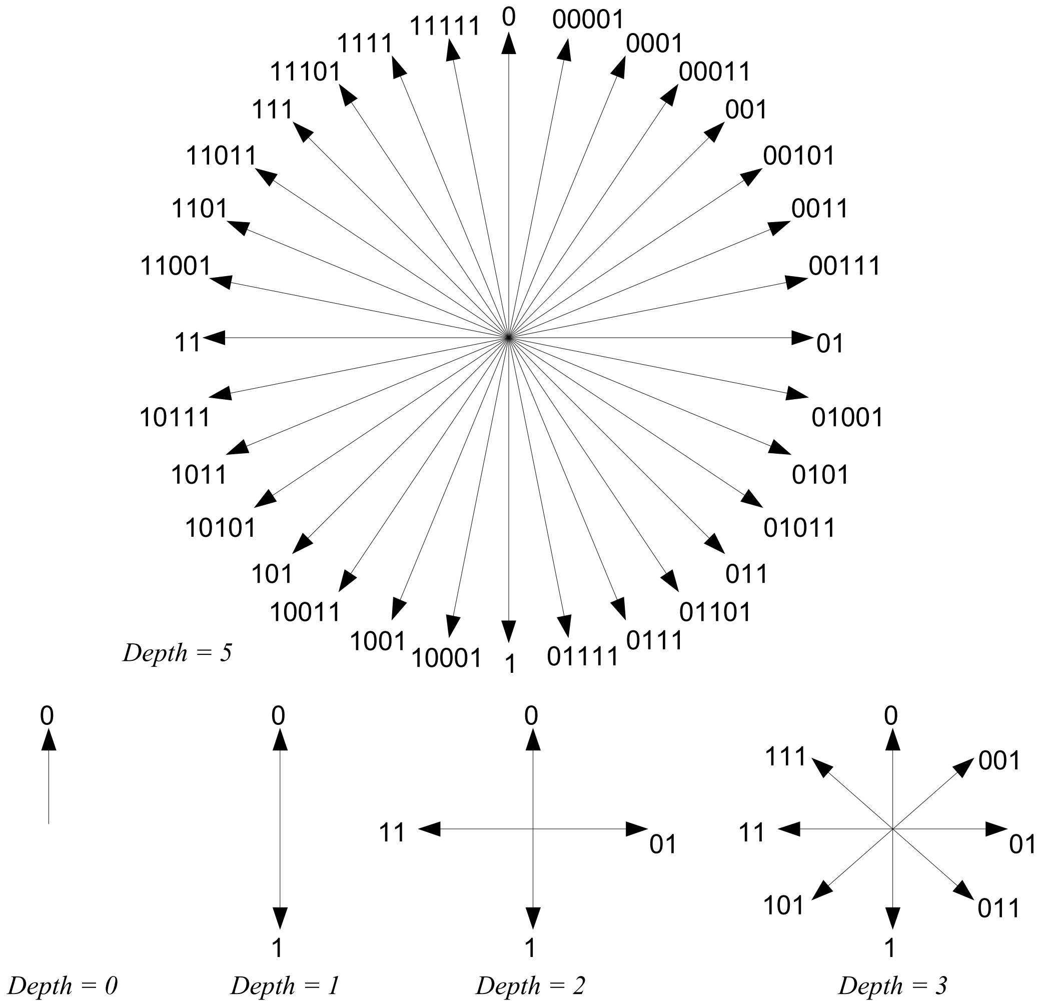


Figure 3: Bunch Direction Codes

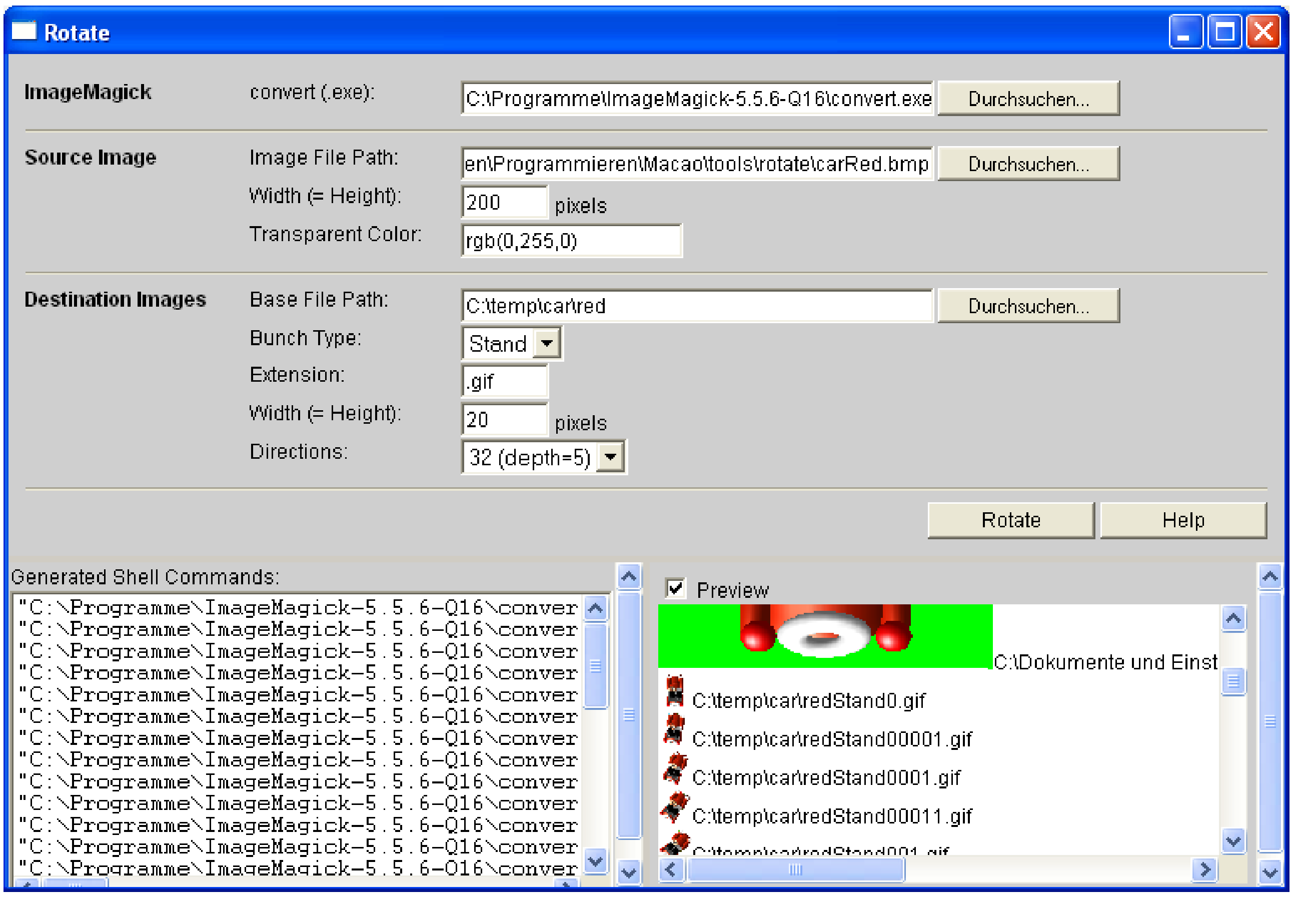
For other objects, which are not shown in birds view, like most characters, you have to draw the images for the different directions yourself. But fortunately you don't have to provide the images for all directions of a bunch. You can omit single directions, if you think they are not really needed. But in this case, you can't use the method addBunchType() to add all images at once. Instead you have to call **createLook()** for each image, providing the correct image name and look name. The format of the look name is also shown above. In addition you have to call the method **setBunchDepth(),** to set the depth of the bunch. Choose a value for the depth, which covers all directions you provide. When the object can't find a fitting image for a direction, it will look for the best fitting image.

The **phaseName** is optional. It is used to switch between different bunches for different situations and/or show a character in different walking phases. Different situations for a car may be for example a loaded and an unloaded car. So you can add bunches for both situations and use the method **setBunchPhase()** to switch between the bunches. Another use for the phaseName is to show a character in different phases while it is walking. The character Sam for example uses three phases to show it walking. One phase is with the left leg ahead and the right leg behind and the second is vice versa. The third phase is with both legs straight like standing. To tell the object to use these phases for walking, you have to call the method **setWalkPhases().** If you used this method to define the walk phases, a phase, which was set using setBunchPhase() is overridden. But you can name walk phases for different situations to combine both effects. So for example you can show a character carrying and not carrying an object while walking.

The **extension** is just the filename extension of all image files. This may be .gif, .png, .jpg or any other image type extension, which the browser supports.

6.4 Rotating Images

To rotate an image of an object, which is shown in birds view, you can use the tool **Rotate**. The tool



ImageMagick
convert (.exe): C:\Programme\ImageMagick-5.5.6-Q16\convert.exe

Source Image
Image File Path: en\Programmieren\Macao\tools\rotate\carRed.bmp
Width (= Height): 200 pixels
Transparent Color: rgb(0,255,0)

Destination Images
Base File Path: C:\temp\car\red
Bunch Type: Stand
Extension: .gif
Width (= Height): 20 pixels
Directions: 32 (depth=5)

Generated Shell Commands:

```
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
"C:\Programme\ImageMagick-5.5.6-Q16\conver
```

C:\Dokumente und Einst
C:\temp\car\redStand0.gif
C:\temp\car\redStand00001.gif
C:\temp\car\redStand0001.gif
C:\temp\car\redStand00011.gif

Figure 4: The tool Rotate

is included in the folder tools/rotate.

Rotate is using the freeware toolset **ImageMagick** to rotate the images. You have to download ImageMagick from the web (http://www.imagemagick.org/) and install it on your computer. You should also download ImageMagicks separate library for gif compression.

Rotate is contained as **rotate.hta** and as **rotate.html** in the distribution of Macao. If you are using a MS Windows system (others see below), start rotate.hta. It is an HTML-application and runs on Windows systems without installation. The application consists of a handful of little HTML- and Java Script files. When you start the application, it offers a dialog like window. Fill out the fields.

Field **ImageMagick - convert (.exe):**	Select the location of the **convert.exe** file of the ImageMagick installation. This file may have another extension, if you are using another operating system.
Field **Source Image - Image File Path:**	Select the source image you want to rotate. The source image has to be of square format. It may be much larger than the resulting bunch images. It is recommended to use a larger source image, because the results will be better. The source image has to show the object in the direction north (= up). The directory tools/rotate contains two example images.
Field **Source Image – Width:**	Enter the size of the source image in pixels. Because the image has to be of square shape, the width is equal to the height.
Field **Source Image – Transparent Color:**	Enter the color code of the background color of the source image. This will be set to transparent in the destination images. You need to know the transparent color. When you open Rotate, there is an example in the field, showing the RGB-color definition for the color green. See the documentation of ImageMagick to find other forms of defining a color.
Field **Destination Images – Base File Path:**	Enter the path, where the bunch images are to be created. Add the beginning of the filename to the path
Field **Destination Images – Bunch Type:**	Select a bunch type. The bunch type will be added to the file name of the created images.
Field **Destination Images – Extension:**	Enter the extension for the created files. ImageMagick will use the extension to identify the encoding of the images. It's recommended to use **.gif** or **.png**-images, because this support a transparent background. You may add a **phaseName** before the dot of the extension. This will let you create images for different phases.
Field **Destination Images – Width:**	Enter the edge length of for the destination images.
Field **Destination Images – Directions:**	Select the number of directions, for which images are to be created. The selection list also shows the corresponding depth for the number of directions.
Field **Preview**:	Activate preview, if you want to see the generated images in the preview frame.

Button **Rotate**:	Click the button Rotate to start the generation of the rotated images.
Generated Shell Commands:	This list shows the generated shell (DOS) commands, which are used to create the images.

The vehicle, which is shown in the source image, should be a little smaller than the image, because when the image is rotated, the corners will be cut.

When you are not using MS Windows, you can't start rotate.hta, because it is M$ specific. But you can open the web page **rotate.html** with your browser. Fill out the fields as described above. Click the button Rotate, to generate the shell commands. Copy the **Generated Shell Commands** to the clipboard and paste them to a shell script file. Execute the shell script file, to generate the images.

6.5 Using the Road Editor

Once you have added the road grid to your page, you can open the **Road Editor** by keyboard. Click on the page and hit 'e'. This will open the road editor window, if the Design Mode is active. Resize the page window and place the Road Editor window beside it.

Note The Design Mode is active by default. See chapter 12.5 on how to deactivate the Design Mode.

Note In the Opera browser, you need to use a menu command to put the Road Editor beside your page. Click with the right mouse button on the tab of your page. Select **Arrange – Tile vertically** to arrange the pages.

You can also open the Road Editor by calling the method **MacaoRoadGrid.openEditor()**.

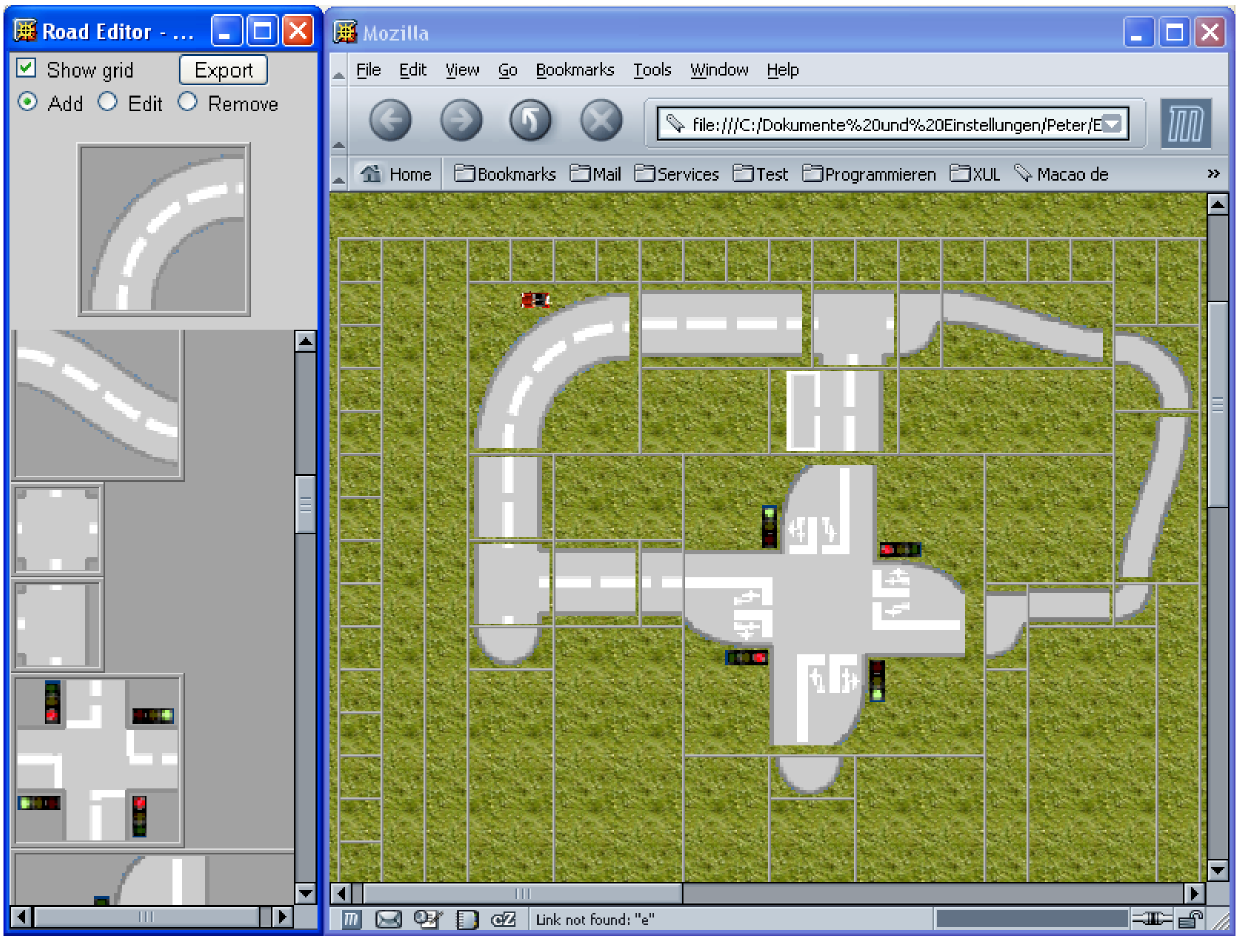
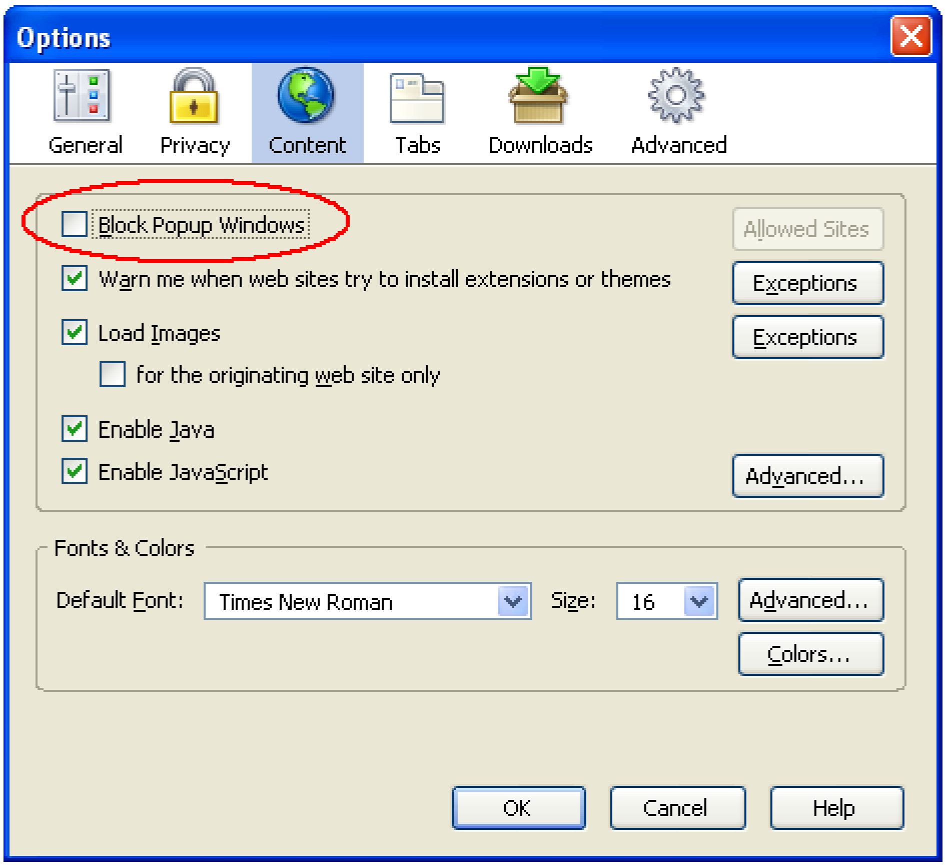


Figure 5: The Macao Road Editor

6.5.1 Deactivate Popup Blockers

Because the Road Editor is opened as an extra window, a **popup blocker** may prevent the opening of the window. If the Road Editor doesn't open, please disable your popup blocker.

If you work with Firefox, the easiest way is to deactivate the entire popup blocker. There is no known way to add the local file system to the list of domains, which are allowed to open popups. Figure 6 shows the deactivation of the popup blocker in the **Firefox** options.



☐ Block Popup Windows
☑ Warn me when web sites try to install extensions or themes
☑ Load Images
☐ for the originating web site only
☑ Enable Java
☑ Enable JavaScript

Default Font: Times New Roman
Size: 16

Figure 6: Deactivation of the Firefox Popup Blocker

6.5.2 Adding a Road Element

First you should activate the checkbox **Show grid**, to show the borders of the grid cells. The grid is optimized, so that it works with a minimum number of cells.

Note The borders of the grid cells are displayed in white color. Change the background of your page to another color or use a background image to make the cell borders visible.

Note The display of the grid uses a little space between the grid cells. So the positions of the road elements will differ from the display without grid lines. Hide the grid to check the right position.

Switch to the add-mode by selecting the radio button **Add**. Select the road type you want to add, by clicking on an image in the list of road types. The selected road type will appear in the frame below the controls of the Road Editor. Click on the displayed road element to rotate the element. Click to the grid position, where the upper left cell of the new road element has to appear, to add the new element.

If there are other road elements, where you tried to place the new element, you will get the error message: **There is not enough space for this element.** Delete the other elements or place the new element at another position.

Note The Road Editor only changes the road elements that are displayed by the road editor. It doesn't re-link the net. So the vehicles can't use the new roads, until you export the generated source code and paste it to the source code of your page.

Note You must not use the first row and the first column of the road grid to place road elements at. The HTML-table may be confused by the colspan or rowspan of these cells, when calculating the cell sizes.

6.5.3 Avoiding Dead Ends

Connections in a MacaoNet are directed. That means, if a node A is connected to B, but not B to A, a vehicle can "walk" from A to B but not back. The road elements in the package optional/road/roadTypes.js are designed for right-hand traffic. On a two- or four-lane road element, the lanes are not connected to each other. But at the junctions there are short cuts, so the vehicles can turn around. There is a termination element for two lane roads, which you can use to terminate a road with a shortcut.

Figure 7: The Termination Road Element

The one lane road elements are so designed, that they use the same nodes for both directions. So a vehicle can turn around at each node. If you are not sure, if your road map has no dead ends, bind an object to the road and call it's method MacaoObject.wanderAround() to let it check out your road map.

6.5.4 Naming a Road Element

To edit the name of a road element, select the mode **Edit** by the radio button. Click on the road element which name you want to change. An input dialog will be opened. If there has not been a name specified for this element, the standard name is displayed. A standard name is assigned to each road element automatically. Enter your specific name and hit OK.

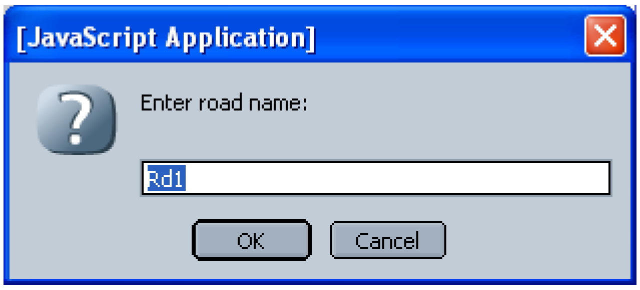


Figure 8: Naming a Road Element

It is useful to name road elements, whose nodes properties you want to change. So you can reach the road element and it's nodes in your program without changing the code, which is generated by the Road Editor. For an example see chapter 6.7 Assigning the Building

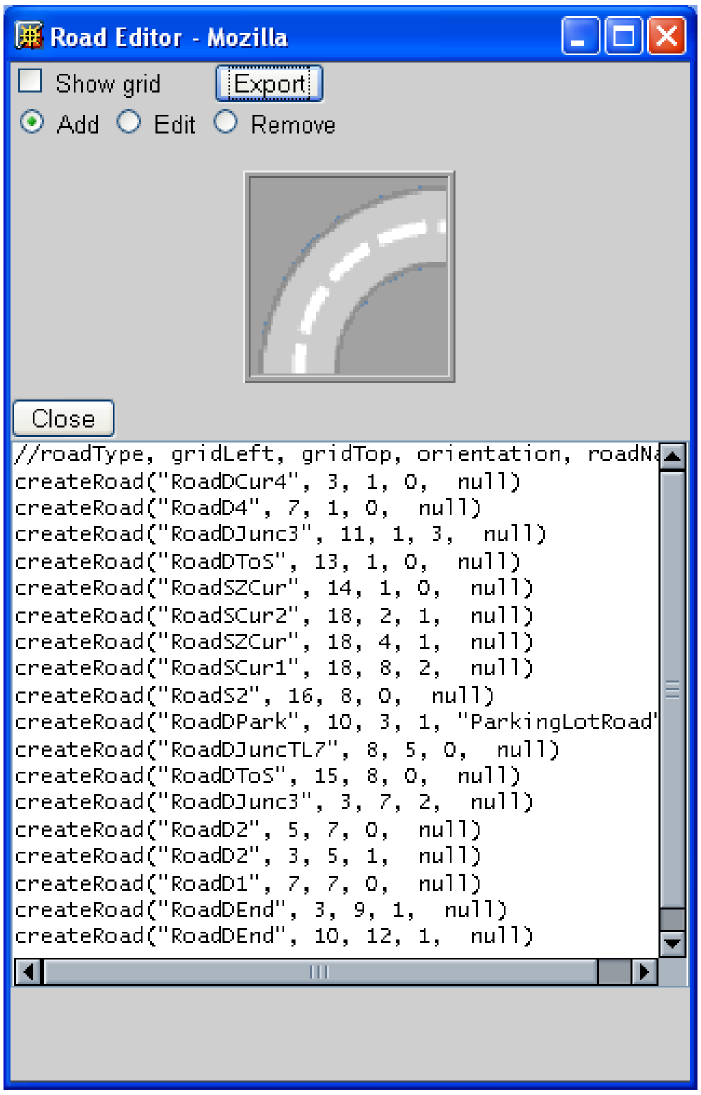
6.5.5 Removing a Road Element

To remove a road element, select the mode **Remove** by the radio button. Click on the road element in your page, to remove the element.

6.5.6 Exporting and Pasting the Road Definition

To put the changes of the road grid permanently to your page, you have to export them from the road editor and paste them to the source code of your page. Click **Export** to display a textbox with the generated source code of the road definition. Select all the text in the text box. Copy the text to the clipboard. Open the source code of your page in an editor. Replace the road definition of your page with the definition in the clipboard. Save the source code and reload the page in the browser. You have to reopen the Road Editor, to use it with the reloaded page.

Note You can reload a single frame in a frameset, if you right click beside the road grid and all other objects of the page and select Reload from the context menu.



```
//roadType, gridLeft, gridTop, orientation, roadNa
createRoad("RoadDCur4", 3, 1, 0,  null)
createRoad("RoadD4", 7, 1, 0,  null)
createRoad("RoadDJunc3", 11, 1, 3,  null)
createRoad("RoadDToS", 13, 1, 0,  null)
createRoad("RoadSZCur", 14, 1, 0,  null)
createRoad("RoadSCur2", 18, 2, 1,  null)
createRoad("RoadSZCur", 18, 4, 1,  null)
createRoad("RoadSCur1", 18, 8, 2,  null)
createRoad("RoadS2", 16, 8, 0,  null)
createRoad("RoadDPark", 10, 3, 1, "ParkingLotRoad"
createRoad("RoadDJuncTL7", 8, 5, 0,  null)
createRoad("RoadDToS", 15, 8, 0,  null)
createRoad("RoadDJunc3", 3, 7, 2,  null)
createRoad("RoadD2", 5, 7, 0,  null)
createRoad("RoadD2", 3, 5, 1,  null)
createRoad("RoadD1", 7, 7, 0,  null)
createRoad("RoadDEnd", 3, 9, 1,  null)
createRoad("RoadDEnd", 10, 12, 1,  null)
```

Figure 9: Road Editor Export

6.6 Adding a Building

Now we add a building to the page. The building is also added using a MacaoObject.

Add the building to the page after the road grid. So the building gets a z-index, which places it in front of the road grid. You can also use the method **MacaoObject.setZIndex()** to manipulate the z-index of the objects.

```
// create and position the house object
var house = new MacaoObject("House1", "House", 174, 260)

// set the image of the house
house.createLook("Default", "optional/landscape/buildings/house7.gif", 128,
135)

// shrink the house image
house.setZoom(0.8)
```

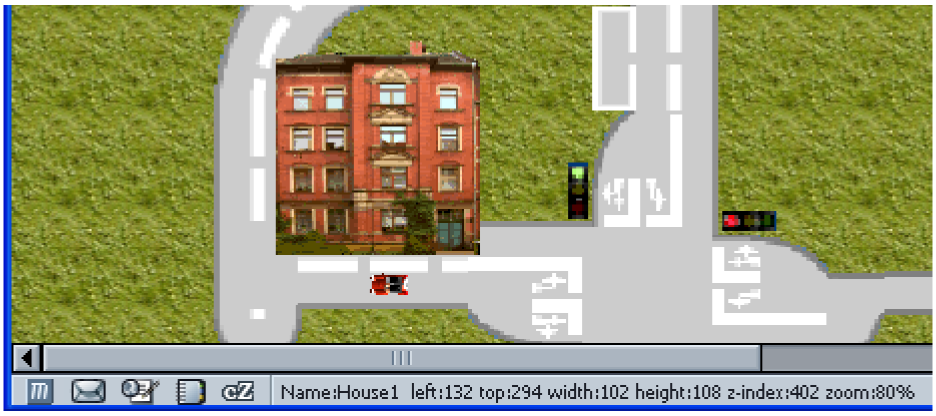

Name:House1 left:132 top:294 width:102 height:108 z-index:402 zoom:80%

Figure 10: The Object Properties in the Status Bar

After you have added the object, you can open the page to display it. You can use the mouse to position the building at the page. Hold down the **shift-** and the **control-key**. Click at the building and hold down the left mouse button. Now you can drag the object across the page. Place it to the right position near the parking lot. While you drag the building, its coordinates are shown in the status bar. Read the left and top coordinates of the object from the status bar and enter them to the source code of your page.

Note In newer versions of the browsers the status bar may be hidden by default. Select **View – Status Bar** or a similar menu item, to make the status bar visible.

Note In the Firefox browser by default the page is not allowed to set the text in the status bar. Execute the following steps to activate the coordinates display in the status bar: Select **Tools – Options** and switch to the **Content** tab. Click on the **Advanced** button for JavaScript options. Activate the checkbox **Change status bar text**.

6.7 Assigning the Building

We will assign the building to that node of the net, which represents the parking lot. So when the user clicks to the house, the car will drive to the parking lot. When the car enters the parking lot, we will use a talk item to react to that event and open another page.

First we have to name the road element of the parking lot. Because a road element consists of a number of nodes and connections, we next have to name the node of the parking lot. Then we will assign the house to the named node.

Use the **Road Editor** to name the road element (See chapter 6.5.4 Naming a Road Element) or enter the name right into the generated source code. The fifth parameter in the line below defines the name of the road element as "ParkingLotRoad".

```
createRoad("RoadDPark", 10, 3, 1, "ParkingLotRoad")
```

To name the node of the parking lot, we get the road element by name and get the node by its index, which is 5. Then we assign a name to the node.

Then we assign the house, which we created above, to the node.

```
// name the node of the parking lot
getRoadByName("ParkingLotRoad").getNodeByIndex(5).setName("ParkingLotNode")

// assign the house to the parking lot node
house.setAssignedNode(getNodeByName("ParkingLotNode"))
```

Now when the user clicks on the house the car will drive to the parking lot.

6.8 Opening another Page

When the car enters the parking lot, it receives the trigger event “enteringNode_ParkingLotNode”. We are adding a talk item to the car to react to this trigger event.

First we let the car say something, when it reaches the parking lot.

```
// let car talk when entering the parking lot
var talkItem = car.createTalkItem(
     "Car_EnteringParkingLot", "enteringNode_ParkingLotNode",
     "I reached the parking lot."
)
```

In this example “enteringNode_ParkingLotNode” is the trigger event, which is generated by Macao.

“Car_EnteringParkingLot” is the name we give our new talk item. This name can be used as trigger for any other talk item of any object. This name has to be unique at least at the page. It is recommended to start the name of the talk item with the name of the object to which it belongs.

“I reached the parking lot.” is the text, which is displayed in the bubble over the car. This parameter is optional. So you can omit it or set it to null, to don't display a bubble but use a talk item for another purpose like sending an event.

We define another event, which is sent, when the talk bubble disappears. We use this event to open the page pub.html, which we will define in the next chapter.

Sending an event is like calling a method. As event name we provide the name of the method to call. In this case it is “openPage”. Second we provide the parameters. The parameters must be enclosed in an array. The link to the page has to be relative to the Macao directory. Third we have to provide the name of the target object. Because the method openPage() is a method of the object, to which the talk item belongs, we just can provide null.

```
// send an event to open the pub page
talkItem.setAfterEvent("openPage", ["examples/tutorial/pub.html"], null)
```

An event has some benefits over just calling a method:

- We can predefine an event.

- An event can be sent to any object in any frame just by providing the name of the target object. The event handling system looks for the recipient object and delivers the event. If the recipient is not there, the event vanishes causing no error message.
- An event can be broadcasted to all objects with one call.
- The delivery of an event can be delayed by posting the event.
- When you provide a target frame name and the page in the target frame is not fully loaded, the event is buffered and delivered, when the page is loaded.

So you should not keep a reference to a target object to call its method. You should better keep the target objects name and send an event to it.

Use the methods **setBeforeEvent()** and **setAfterEvent()** to define the events of a talk item. You can also use the methods **sendEvent()**, **postEvent()** and **broadcastEvent()** of an object or of the page to send events.

6.9 Adjusting the Controllers

We included the controller page in one of the frames. The car can be steered without the steering controller. But we activate the steering controller, so the user has an additional way to control the car.

The steering controller has got the name “SteeringController” in the controller page. First we assign this controller to the car. So it will listen to the controllers' commands.

```
// link the car to the steering controller
car.addController("SteeringController")
```

Next we tell the controller page, which controllers are needed for our roadmap page.

The controller page consists of the three controllers “**TalkController**”, “**SteeringController**”, “**SententceController**” and the “**ControllerMenu**”. The ControllerMenu appears at the left side and is used to switch between the controllers. The user can use the Controller Menu to switch between the controllers, if there is more than one controller available.

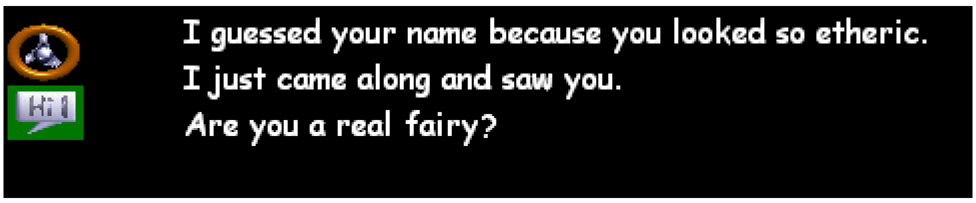

I guessed your name because you looked so etheric.
I just came along and saw you.
Are you a real fairy?

Figure 11 : Controller Menu and Talk Controller

The **TalkController** makes itself visible, if it has to display a list of answers to the user. When the user selects one of the answers, the TalkController hides it self and makes the default controller active. So it needs to know, which is the default controller.

The following code first defines, which of the three controllers are available to the user via the menu, when the page is opened. This may be more than one. The fourth command tells the TalkController which is the default controller. The last command tells the menu, which controller is to be shown initially.

```
// hide talk controller
sendEvent("setMenuItemVisibility", ["TalkController", false],
     "ControllerMenu", "controllers")

// show steering controller
sendEvent("setMenuItemVisibility", ["SteeringController", true],
     "ControllerMenu", "controllers")

// hide sentence controller
sendEvent("setMenuItemVisibility", ["SentenceController", false],
     "ControllerMenu", "controllers")

// make the steering controller the default controller
sendEvent("setDefaultControllerMenuItem", ["SteeringController"],
     "TalkController", "controllers")

// switch to steering controller
sendEvent("setMenuItemActive", ["SteeringController"],
     "ControllerMenu", "controllers")
```

The objects at the controller page are changed by sending events. Four events are sent to the **ControllerMenu** and one is sent to the TalkController object. To make sure the commands are delivered, even if the controller page is not fully loaded, the name of the frame “controllers” is added. This forces the event system to buffer the command for that frame.

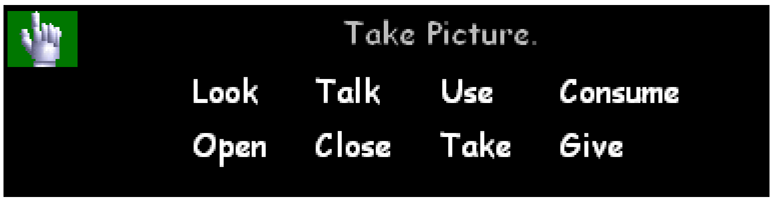


Figure 12: The Sentence Controller.

6.10 Using the Steering Controller

The user can use the steering wheel of the **SteeringController** to predefine, which connection is to be used at the next forking. The **steering wheel** can be turned ahead, left, sharp left, right and sharp right. When the next node is passed, the car follows the predefined path and the steering wheel is adjusted.

The **accelerator stick** shows the current velocity of the steered car. The user can start the car by clicking at the steering wheel. Clicking at the accelerator stick stops the car.

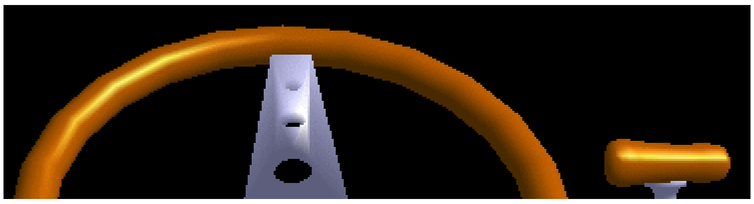


Figure 13: The Steering Controller

6.11 Adding a Horizon

A nice decoration at a road map page can be a horizon. The horizon uses no features of Macao. But it shows how you can combine Macao with static **HTML**.

There are three horizon images provided with the Macao distribution, which can be used for that purpose. Of course you can make your own. The images are faded at the bottom edge to the grass tile, which is used as background for the rest of the page.

The horizon is build by using the horizon picture as repeating background of a table cell. So the horizon is as wide and height as the table cell. The size of the only table cell is set by a transparent gif image, which is stretched to the required size.

```
<body
     background="../../optional/landscape/background/grasTile.gif"
     bgcolor="gray"
>
<table
     background="../../optional/landscape/horizon/horizon1.jpg"
     border=0 cellpadding=0 cellspacing=0
>
     <tr>
           <td>
                 <image
                       src="../../core/1p.gif"
                       width="1830" height="195" border="0"
                 >
           </td>
     </tr>
</table>
<!-- ... -->
</body>
```

You should adjust the width of the table cell to the width of your page. The height should fit exactly the horizon image height.

6.12 Positioning the Car

Now we are making the start position of the car dependent, if the user is starting our application or if the pub page opens the road map page. If the user comes from the pub page, the car is to be positioned at the parking lot.

With the function **getOpenerPageName()** we can get the name of the page, which opened the current page. This does only work, if the page is opened by one of the **openPage()** methods. These methods are adding the page name as parameter to the URL. The method returns the file name of the opener page without extension.

To find the right position for your car, you can hold down the Shift and the Control key. Then you can drag and drop the car as any object to the right position. The coordinates of the object are displayed in the Status Bar of the browser.

When the pub page opens the roadmap page, we use the method **moveToNode()**. With this method we need not to figure out the node position. This method makes the ParkingLotNode the actual node of the car, without firing the event enteringNode. Firing the event is prevented by the fourth parameter. If the event were fired, the pub page would be opened again.

```
// is opened from pub page
if(getOpenerPageName() == "pub") {

     // position car at parking lot face down
     car.moveToNode(getNodeByName("ParkingLotNode"), 0, 1, true)
} else {

     // position car on road
     car.moveTo(201, 250)

     // face right
     car.setDirection(1, 0)
}
```

7 Creating a Page Showing a Room

In this chapter we will create a page, which is showing a room. The room is a pub. We will put some furniture to the room. Then we create two characters, which are Sam and the Barkeeper. We will learn, how to animate the characters and how to create the paths to make the characters walk on. We will learn how to stack the objects at the page and how to zoom Sam, when he is walking to the background.

You can find the example of this chapter under examples/tutorial. Start the page index.html and drive with the car to the parking lot to enter the pub. The pub is implemented in the page pub.html.

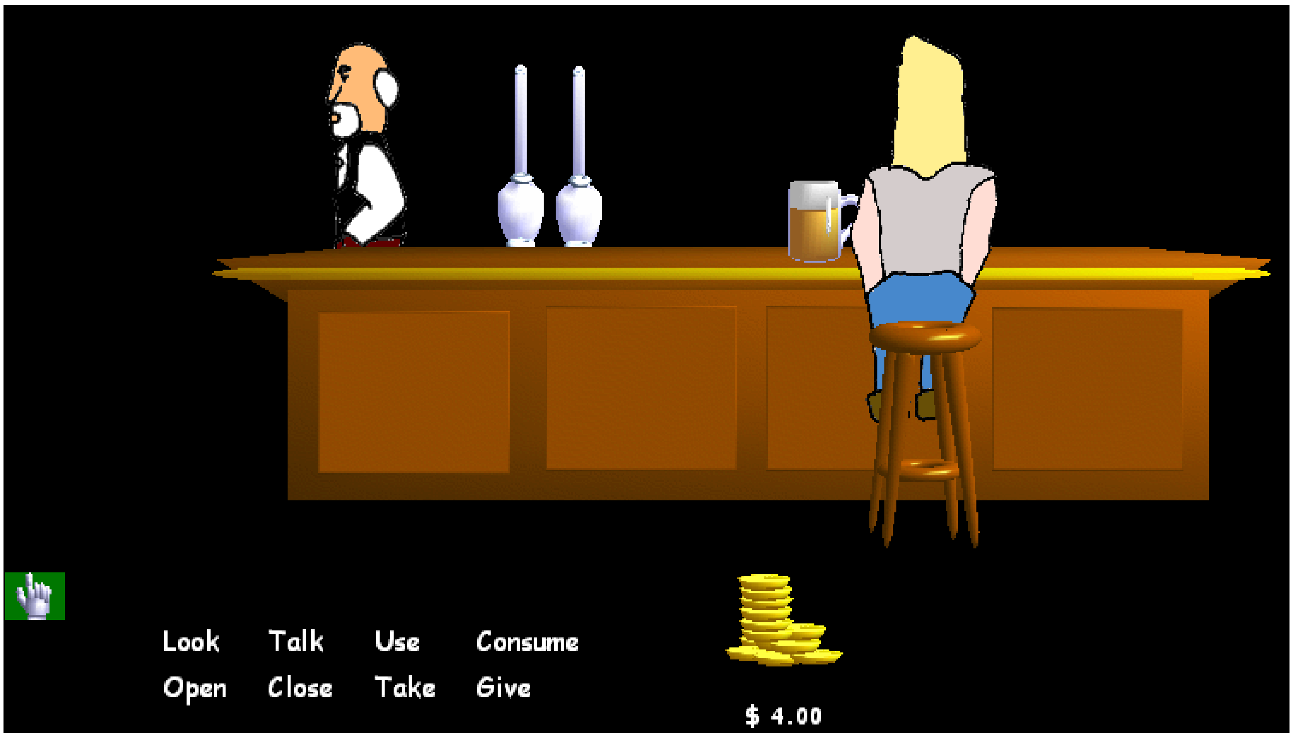


Figure 14: The Pub Page

7.1 Creating the Pub Page

First we have to create a new HTML-page. This page is similar to the road page, but it doesn't need to load the road libraries.

```
<html>
<head>
<meta http-equiv="expires" content="10000">
<script>
var basePath = "../../"
</script>
<link rel=stylesheet type="text/css" href="../../core/macao.css">
</head>
<body style="background-color: black;">
<script language="JavaScript" src="../../core/kernel.js"
type="text/javascript"></script>
<script language="JavaScript" src="../../core/dynamic.js"
type="text/javascript"></script>
<script language="JavaScript" type="text/javascript">

// ... your code goes here

</script>
</body>
</html>
```

Note To bring a better structure to the code, the example application is split into some functions. All the functions are called by the function createAllObjects(), which is called once, when the page is loaded. These functions are not shown in the tutorial, because they are quite forward and are not necessary to understand the Macao API.

7.2 Setting the controllers

Now we activate the controllers, which are useful for a room page.

We hide the **SteeringController**, which is only useful at pages with a roadmap.

We hide the **TalkController**, which will be displayed automatically, when a list of answers is to be displayed.

We show the **SentenceController**. The user can use the SentenceController to build a command sentence by clicking objects at the page or at the inventory. The built sentence is send as trigger event to every object. We will later add TalkItems, which are triggered by such trigger events.

We switch the **ControllerMenu** to the SentenceController menu item, so it fits the visible controller.

At last we tell the TalkController to make the SentenceController visible, when the TalkController hides it self.

```
// hide steering controller
sendEvent(
     "setMenuItemVisibility", ["SteeringController", false],
     "ControllerMenu", "controllers"
)

// hide talk controller
sendEvent(
     "setMenuItemVisibility", ["TalkController", false],
     "ControllerMenu", "controllers"
)

// show sentence controller
sendEvent(
     "setMenuItemVisibility", ["SentenceController", true],
     "ControllerMenu", "controllers"
)

// switch to sentence controller
sendEvent(
     "setMenuItemActive", ["SentenceController"],
     "ControllerMenu", "controllers"
)

// make the sentence controller the default controller
sendEvent(
     "setDefaultControllerMenuItem", ["SentenceController"],
     "TalkController", "controllers"
)
```

7.3 Adding Static Objects

Now we add a few static objects to the pub. There is a bar counter and a beer tap at the counter. In front of the counter stands a barstool. Each object gets a single look, which can be named with any name.

Again you can figure out a fitting position at the page, by dragging the objects with the mouse, while holding the Shift and the Control button down.

We hide the **tool tip** of the counter. The tool tip is initially set to the value of the title of the object. A tool tip of a large object like the counter may confuse the user.

```
var counter
var tap
var barstool

// create bar counter
counter = new MacaoObject("Counter", "Counter", 123, 187)

// add the only look
counter.createLook("Default", "optional/room/bar/barCounter.gif", 619, 148)

// remove the tool tip
counter.setToolTip(null)


// create barstool
barstool = new MacaoObject("Barstool", "Barstool", 507, 230)

// add the only look
barstool.createLook("Default", "optional/room/bar/barstool.gif", 66, 133)


// create tap
tap = new MacaoObject("Tap", "Tap", 290, 80)

// add the only look
```

```
tap.createLook("Default", "optional/room/bar/tap.gif", 61, 112)
```

7.4 Stacking Objects

Stacking is used to put the objects one over the other. So we want to have the barstool in front of the counter.

We can control the objects stacking by setting the **z-index**. Or we let Macao adjust the z-index. When Macao calculates the stacking, it sets the z-index to the value of the y-position of the lower edge of the object. So objects, which are positioned lower at the page, are appearing in front of objects, which are placed more to the top of the page.

This works as long, as all objects are standing on a common virtual floor. In our example, the counter and the barstool are standing on a “common floor”. We place the barstool a little lower at the page and call **adjustStacking()** to both objects, so the barstool appears in front of the counter.

We position the tap on top of the counter. We want the tap to appear in front of the counter. But the bottom edge of the tap is more to the top of the page, than the bottom edge of the counter. So the stacking of Macao would place it behind the counter. To solve this problem we just assign the z-index of the tap directly. It is calculated by the y-value of the bottom edge of the counter plus 1.

```
// automatic calculation of z-index of counter
counter.adjustStacking()

// automatic calculation of z-index of barstool
barstool.adjustStacking()

// manual setting of z-index for tap
tap.setZIndex(334)
```

The stacking of walking objects is adjusted automatically while walking. You can call the method **setStacking()** to activate or deactivate this behavior.

If you like to have another calculation of the z-index, you can override the method **calculateStackingIndex()** of an object. An overriding could be done like this:

```
sam.calculateStackingIndex = function() {

     // calculate from bottom position
     return this.getTop() + this.getHeight()
}
```

This overriding just implements the same calculation like the standard behavior.

7.5 Adding the Character Sam

We are adding the character Sam to the pub. We use Sam as our hero, which will be controlled by the user.

To let the user interact with Sam, we assign Sam to the **TalkController** and the **SentenceController** .

```
var sam

// create sam
sam = new MacaoObject("Sam", "Sam", 54, 224)

// register controllers
sam.addController("TalkController")
sam.addController("SentenceController")
```

7.6 Animating the Character

Because a character looks different when standing and walking, we add a **look bunch** for standing and a look bunch for walking.

The look bunch for walking consists of eight directions (directions = 2^{depth}) and of three images for each direction. These images per direction, which are called **phases**, are showing Sam with his legs in different positions. We name this phases “a”, “b” and “c”. The phase names are also part of the image filenames. With the ninth parameter of the method **addLookBunch()** we tell the object to create a look for each phase.

The call of **setWalkPhases()** tells the object how to use these phases for walking. The object will show another look for each step while walking in the given order: a, b, c, b, a, b, c,...

```
// define look for walking
sam.addLookBunch(
     sam.BUNCH_TYPE_WALK,
     "optional/characters/sam/phases/Sam", 3, null, 90, 230, 0, 0, "a,b,c"
)

// define the walk sequence
sam.setWalkPhases("a,b,c,b")

// define look for standing
sam.addLookBunch(
     sam.BUNCH_TYPE_STAND,
     "optional/characters/sam/Sam", 3, null, 90, 230
)
```

Note You can use phases also to show an object in different situations. Do this by adding more phases to the object and change the walk phases during the game. Or call setWalkPhases(null) and use **setBunchPhase()** to select an explicit phase.

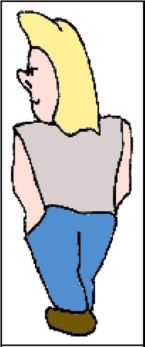
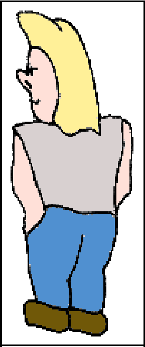
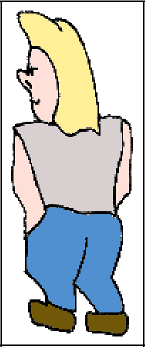
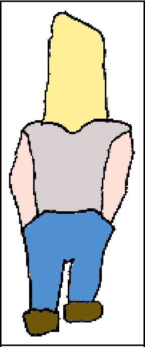
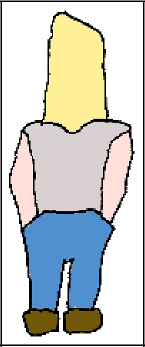
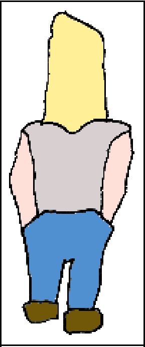
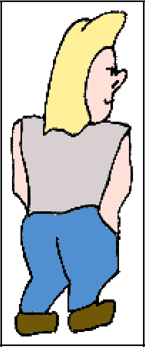
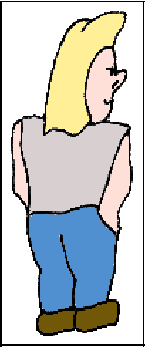
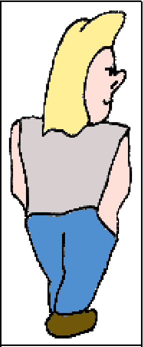
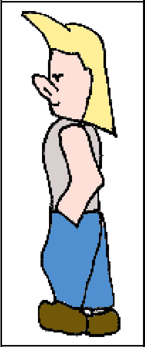
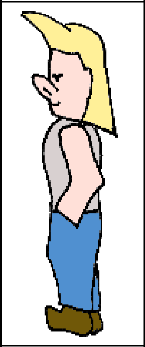
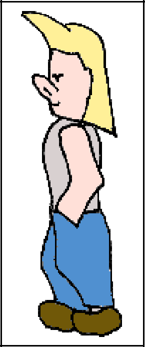
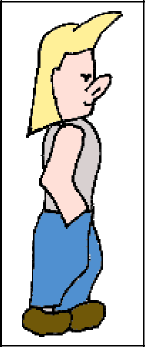
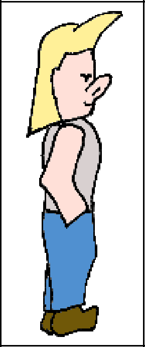
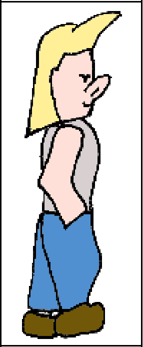
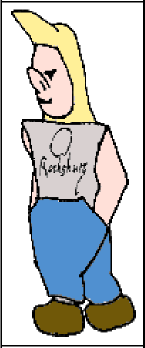
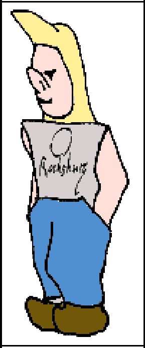
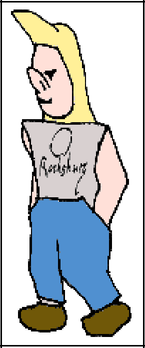
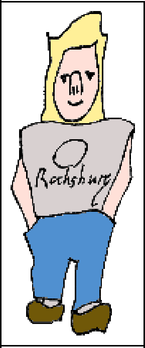
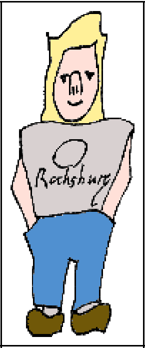
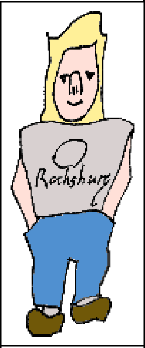
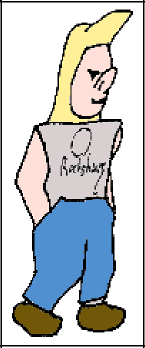
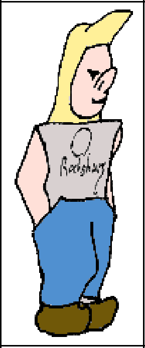
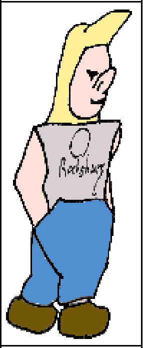
								
SamWalk 111a.gif	SamWalk 111b.gif	SamWalk 111c.gif	SamWalk 0a.gif	SamWalk 0b.gif	SamWalk 0c.gif	SamWalk 001a.gif	SamWalk 001b.gif	SamWalk 001c.gif
								
SamWalk 11a.gif	SamWalk 11b.gif	SamWalk 11c.gif				SamWalk 01a.gif	SamWalk 01b.gif	SamWalk 01c.gif
								
SamWalk 101a.gif	SamWalk 101b.gif	SamWalk 101c.gif	SamWalk 1a.gif	SamWalk 1b.gif	SamWalk 1c.gif	SamWalk 011a.gif	SamWalk 011b.gif	SamWalk 011c.gif

Figure 15: Sam's Walking Phases

In Figure 15 you can see the images for the walking directions and phases of the character Sam. You don't need as much images to create a new walking character. The character Doc for example has only images for four directions while Sam has images for eight directions. There is also much copy and paste in the creation of the images.

Now we set the initial direction of Sam. With x=1 and y=-1 he is looking northeast. We let Sam approach the users mouse click position. If Sam leaves the visible area of the page, we make the page scroll automatically.

```
// set initial direction
sam.setDirection(1, -1)

// make the charackter walk to the mouse click position
sam.setApproachClick(true)

// make the page scroll
```

```
sam.setScrollVisibleOptions(true, 20)
```

We change the maximum **curve velocity** to 1.5, which is about PI/2. This reduces the curve radius, when Sam walks. A curve velocity of PI/2 means, Sam can walk a 360-degree turnaround in four steps. The default value is 0.62, which is more fitting for cars. When the curve radius is too large, the object may walk too far away from the nodes and connections and might walk through furniture.

```
// reduce curve radius
sam.setWalkCharacteristics(null, null, 1.5, null)
```

When the user wants to click at an item on the page, Sam might cover this item with his body. To let the user click at the object, we make Sam transparent for mouse events. So when the user clicks at Sam, the mouse event is forwarded to the object behind Sam.

In addition, we remove the tool tip from Sam.

Note Currently there is no way implemented to activate the **tool tip** of another object, while Sam stands in front of it.

```
// make transparent for mouse events
sam.setForwardMouseEvents(true)

// remove the tool tip
sam.setToolTip(null)
```

7.7 Defining Paths

We create a net for Sam to walk on. We name this net “SamsNet”. We create five nodes. The nodeEntrance is left in the foreground, where Sam will enter the pub. The nodes nodeFront and nodeRight are left and right in front of the counter. The nodeBarstool is left beside the barstool. The nodeBack is right in the background. This Node will be used to show zooming.

We give each node a name. So Sam will receive trigger events, when he enters and leaves these nodes. We will use these trigger events to trigger talk items to control Sam. We can also obtain these nodes by their name. Make sure you use unique names for the nodes of all nets at a page. We will later create a second net for the barkeeper.

```
var netName
var nodeEntrance
var nodeFront
var nodeBarstool
var nodeRight
var nodeBack

// define the name of sam's net
netName = "SamsNet"

// create node at entrance
nodeEntrance = new MacaoNode(netName, 98, 475, null, "SamEntrance")
```

```
// create node in front of counter
nodeFront = new MacaoNode(netName, 360, 404, null, "SamFront")

// create node beside barstool
nodeBarstool = new MacaoNode(netName, 455, 357, null, "SamBarstool")

// create node right in front of counter
nodeRight = new MacaoNode(netName, 724, 413, null, "SamRight")

// create node in the back
nodeBack = new MacaoNode(netName, 874, 301, null, "SamBack")
```

Next we link the nodes together. Make sure to provide true for the second parameter. This will link the nodes also in the opposite direction. Else you would create a one-way street. The nodeFront, nodeRight and nodeBarstool are building a triangle.

```
// link nodes
nodeEntrance.connectTo(nodeFront, true)
nodeFront.connectTo(nodeBarstool, true)
nodeFront.connectTo(nodeRight, true)
nodeRight.connectTo(nodeBarstool, true)
nodeRight.connectTo(nodeBack, true)
```

Now we are binding Sam to his net. So he will use this net for walking.

We move Sam's **anchor point** to his bottom edge. The anchor is a virtual point, which is used to let the object walk along the net. By default the anchor is at the center of the object. This is fitting for cars but not for characters.

We place him at the entrance node when the page is loaded. He looks x=1 and y=-1 which is northeast. We set the fourth parameter to true, so Sam will receive no trigger events during this move to the node. We will later use this trigger event to make Sam leave the pub, when he enters the node again.

```
// bind sam to the net
sam.bindToNet(netName)

// change the anchor position
sam.setAnchorAtBottom(true)

// set position and direction
sam.moveToNode(nodeEntrance, 1, -1, true)
```

7.8 Zooming Objects

We will make Sam change his size while walking along the net. We do this by assigning a **zoom factor** to each node.

By default the zoom factor of a node is zero, which tells the object not to change its zoom factor. So you don't need to assign a factor to each node. If you want the walking object to change it's zoom factor between two adjacent nodes, just set the zoom factors of this nodes.

Because Sam starts at the entrance node, we set his initial zoom factor to the zoom factor of the entrance node.

```
// set zoom factors
nodeBack.setZoom(0.8)
nodeRight.setZoom(1.0)
nodeFront.setZoom(1.0)
nodeEntrance.setZoom(1.1)
sam.setZoom(1.1)
```

You can also change the size of every static object by setting a zoom factor to it.

Note You can set a basic zoom factor of an object by providing recalculated sizes when creating its Looks. The **baseWith** and **baseHeight** you provide for a look doesn't need to fit the image size.

7.9 Creating the Barkeeper

We create the Barkeeper as a second character at the page. This is similar to the character Sam.

For the Barkeeper we use the images of the character Doc. Because the Barkeeper walks behind the counter, we do not really need the walk-bunch. The walking legs are not visible.

Because user doesn't control the Barkeeper, we don't have to assign the controllers and we don't make him transparent for the mouse clicks.

```
var barkeeper

// create barkeeper
barkeeper = new MacaoObject("Barkeeper", "Barkeeper", 200, 65)

// define look for walking
barkeeper.addLookBunch(
     barkeeper.BUNCH_TYPE_WALK,
     "optional/characters/doc/phases/Doc",
     2, null, 97, 200, 0, 0, "a,b,c"
)
barkeeper.setWalkPhases("a,b,c,b")

// define look for standing
barkeeper.addLookBunch(
     barkeeper.BUNCH_TYPE_STAND,
     "optional/characters/doc/Doc",
     2, null, 97, 200
)

// set initial direction
barkeeper.setDirection(0, 1)

// change the anchor position
barkeeper.setAnchorAtBottom(true)
```

The Barkeeper gets his own net “BarkeeperNet”. It consists of four nodes behind the counter. The nodes are connected in a row.

```
var netName
var nodeLeft
var nodeTap
var nodeCenter
var nodeRight

// define net name
netName = "BarkeeperNet"

// create node at left
nodeLeft = new MacaoNode(netName, 212, 265, null, "BarkeeperNodeLeft")

// create node at tap
nodeTap = new MacaoNode(netName, 312, 265, null, "BarkeeperNodeTap")

// create node at center
nodeCenter = new MacaoNode(netName, 440, 265, null, "BarkeeperNodeCenter")

// create node at right
nodeRight = new MacaoNode(netName, 593, 265, null, "BarkeeperNodeRight")

// connenct nodes
nodeLeft.connectTo(nodeTap, true)
nodeTap.connectTo(nodeCenter, true)
nodeCenter.connectTo(nodeRight, true)

// bind barkeeper to net
barkeeper.bindToNet(netName)
```

We adjust the maximum velocity and the curve radius of the Barkeeper. If the curve radius is too large, the Barkeeper might walk “through” the counter. This would look curious. You define not the radius itself but the angle per walking step in radiant. So 1.5 is about PI/2 per step. This means, the Barkeeper can do a 180-degree turn in two steps. With a maximum velocity of 10 pixels per step, the Barkeeper will probably walk no more than 10 pixels away from the defined net.

The parameters, which are set to null are staying unchanged.

```
// reduce velocity and curve radius
barkeeper.setWalkCharacteristics(10, null, 1.5, null)
```

7.10 Making an Object Wander Around

To animate the scene, we make the Barkeeper wander around. Calling the method **wanderAround()** lets the Barkeeper select a random node of his net and walk to it. Then he waits for a moment and walks to the next node. We provide 3500 milliseconds as base time to calculate the time for waiting.

In addition, you could provide an array with nodes, which the Barkeeper uses as random walk targets.

Later we will have to restart the wanderAround, after the Barkeeper does another kind of walk.

```
// make barkeeper wander around
barkeeper.wanderAround(3500)
```

7.11 Adding a Background

To add a background to a page showing a room, you should create a JPEG image showing the background. Set the image as background to the body tag. Use the style attribute, to set the background to "no-repeat", so the image is not tiled but only displayed once. Set the rest of the background to a fitting color. Set the margin to zero, so the image is aligned to the left top corner of the page.

```
<body
     background="../../optional/room/background/greenOffice.jpg"
     style="
            margin:0px;
           background-color: black;
            background-repeat: no-repeat;
     "
>
...
</body>
```

Note Since the pub page currently has no background image, this example is taken from the psychoffice.html of Discover Macao.

7.12 Setting the Size of the Page

You should explicitly set the size of the page. Else the size might be affected by the position of walking objects. This looks funny but might not be what you want. A background will not affect the size of the page.

So we are placing a tiny DIV at the lower right corner of the page. In the DIV we are putting an image, which is referencing the transparent image **1p.gif** from the core directory. We are using the left and the top rule in a style attribute to position the DIV.

Don't enter whitespace between the DIV tags and the image tag, because this would enlarge the height of the DIV to the height of a text row.

```
<div style="position:absolute; left:1000px; top:415px; with:1px; heigth:1px;"
><img src="../../core/1p.gif" width="1" height="1"
></div>
```

8 Let the Objects Talk

In this chapter we will learn how to use **TalkItems** to predefine the behavior of the objects. We will let Sam use the barstool and order a beer. We will make the Barkeeper bring the beer.

You will learn, how to react to commands from the **SentenceController**. The user can use the SentenceController to build command sentences to control an object.

You will learn, how to use the **TalkController** to let the user talk to characters.

We will use the **Inventory** to let Sam take an item from the scene and to pay his bill.

You can find the example of this chapter under examples/tutorial. Start the page index.html and drive with the car to the parking lot to enter the pub. The pub is implemented in the page pub.html.

8.1 What is a TalkItem?

We will add a number of **TalkItems** to each dynamic object, to define its behavior. TalkItems are defining how an object has to react to trigger events.

A TalkItem can consist of a text and two events.

When the TalkItem is executed, the text is displayed as **bubble** for the talking object.

The events are predefined calls of methods of an object. There is one Before- and one After Event, which are executed before or after the display of the bubble. Each of the elements, the text and the events are optional. There are a lot more properties, which you can set to a TalkItem.

The execution of a TalkItem is triggered by **trigger events**. Trigger events are just strings, which are send to the object and may trigger the execution of a fitting TalkItem. A trigger event can be the name of another TalkItem, which has just been executed. It also can be a user event like a mouse event. You can define one or more trigger events, which are triggering the execution of the same TalkItem.

If there are more than one TalkItem belonging to an object, which are triggered by the same trigger event, the TalkItem to execute is selected randomly from this items.

This is the basic creation of a TalkItem:

```
var talkItem = object.createTalkItem(talkItemName, triggerEvents, message)
```

You must call the method **createTalkItem()** to create a TalkItem and add it to the object.

The first parameter is the TalkItem's name. The name must be unique in the page. It is recommended to start the name of the TalkItem with the name of the object it belongs to.

The second parameter contains the trigger events, which are triggering the execution of this TalkItem. Provide a string, if there is only one trigger event. Provide an array of strings, if there are more trigger events.

The third parameter is the message, which is displayed in the bubble. Provide the message, which the object has to say, when the TalkItem is executed. Provide null, if there is no message to be displayed.

The call of the method creates a **TalkItem** and adds it to the object. It returns the TalkItem. You can assign the returned TalkItem to a variable and set additional properties of that TalkItem by calling it's methods.

8.1.1 Node Trigger Events

The following table lists the trigger events, which an object receives, when it enters or leaves the node of a net.

Trigger Event Name	***Description***
enteringNode_*NodeName*	This event is raised, when the object enters a named node. Replace *NodeName* with the name of the node.
leavingNode_ *NodeName*	This event is raised, when the object leaves a named node. Replace *NodeName* with the name of the node.
leavingNode_*NodeName*_to _*NextNodeName*	This event is raised, when the object leaves one named node in the direction of another named node. Replace *NodeName* with the name of the node, which is left. Replace *NextNodeName* with the name of the next node to which the object is walking.

Table 1: Node Trigger Events

8.1.2 Sentence Controller Trigger Events

The user can use the **SentenceController** to build a sentence by clicking at a command and one or two objects. The built sentence is send as trigger event to each object, which is assigned to the SentenceController. The following table lists the trigger events, which are sent by the SentenceController.

Trigger Event Name	***Description***
Command	This event is raised, when the user clicks one command of the sentence controller. Replace *Command* with one of the commands of the SentenceController.
Command_Object1	This event is raised, when the user clicks one command on the sentence controller and then one object. Replace *Command* with one of the commands of the SentenceController. Replace *Object1* with the name of the object.
Command_Object1_Object2	This event is raised, when the user clicks one command at the sentence controller and then two objects. Replace *Command* with one of the commands of the SentenceController. Replace *Object1* and *Object2* with the names of the objects.
*Command*_Default	This event is raised, when the user has clicked one command on the sentence controller and then one or two objects and there is no fitting TalkItem. So you can define a default TalkItem, which is executed, when the user doesn't find the fitting objects. If the TalkItem is executed after the user has clicked the first or the second object depends on, if there is a conjunction defined for the verb at the SentenceController or is not. If there is no conjunction defined, then the sentence controller decides after the first object, if the user failed to build the valid sentence, and sends the default event. Replace *Command* with one of the commands of the SentenceController.

Table 2: SentenceController Trigger Events

The next table lists the commands, which the SentenceController has by default. You can change this commands by copying and modifying the page optional/controllers/controllers.html.

Command	***Conjunction***
Look	-
Talk	-
Use	with
Consume	with
Open	with
Close	with
Take	-
Give	to

Table 3: SentenceController Commands

“**Look**” is the default command. When the user clicks at an object before clicking a command, the command Look is used to build the sentence.

See the API documentation of the class TalkItem to find more useful trigger events.

8.2 What is an Event?

An event is like a predefined call of a method. To predefine an event, you have to provide the name of the method and the parameters of the call.

If you define the Before- or After Event of a TalkItem, the event is by default sent to the object itself. But you can provide the name of an object to define it as the receiver of the event.

```
talkItem.setBeforeEvent(eventName, parameters, targetObjectName)
talkItem.setAfterEvent(eventName, parameters, targetObjectName)
```

The first parameter is the event name, which is the name of the method, which is to be called by the event. See the **API Documentation** for the available methods of MacaoObjects.

The second parameter gets the parameters for the method call. Because there can be more than one parameter, the parameters must be enclosed in an array. Provide null, if there are no parameters.

The third parameter gets the name of the object, whose method is to be called. Provide the name of an object. Provide null, if the method belongs to the same Object as the TalkItem.

8.3 What are GameEvents?

GameEvents are named flags. You can set and clear these flags.

The GameEvents are common to all pages of your site, which are using the same **StorageManager**. So the status of the game is passed from page to page.

You can activate or deactivate TalkItems referring to GameEvents.

When a TalkItem is executed, it can set and clear GameEvents.

Note If you need to pass more complex information from page to page, you can use the **StorageManager** to store strings and numbers. Use **setStorageValue()** and **getStorageValue()**.

The following call defines the behavior of a TalkItem regarding GameEvents. All parameters are names of GameEvents and each parameter may be null.

```
talkItem.setGameEvents(beforeGameEvent, afterGameEvent, setGameEvent,
clearGameEvent)
```

If *beforeGameEvent* is provided, the TalkItem is not executed, until the provided GameEvent is set.

If *afterGameEvent* is provided, the TalkItem is not executed, after the provided GameEvent is set.

If *setGameEvent* is provided, the provided GameEvent is set, when the TalkItem is executed.

If *clearGameEvent* is provided, the provided GameEvent is cleared, when the TalkItem is executed.

8.4 Sam and the Barstool

We will define a few **TalkItems** to tell Sam how to work with the barstool.

We want the user to build the sentence “Use Barstool” with the SentenceController to sit on the barstool. So we use the trigger event “Use_Barstool” to trigger our first TalkItem. We define the position and the direction, which Sam has to use while sitting on the barstool.

When Sam is sitting down, we use the BeforeEvent to disable the automatic walk to the mouse position. Else Sam would stand up again at the next mouse click.

When Sam sits down, we are setting the Game Event “SamAtBarstool”. So we can make the behavior of objects dependent on this situation. The first dependency is this TalkItem itself. When Sam is already sitting, he can't sit down again.

```
var talkItem

// - enter barstool -
talkItem = sam.createTalkItem("Sam_EnterBarstool", "Use_Barstool")

// only if not at barstool, set game event
talkItem.setGameEvents(null, "SamAtBarstool", "SamAtBarstool", null)

// define position and direction on barstool
talkItem.setStartPosition(540, 290)
talkItem.setStartDirection(0, -1)

// don't walk while on barstool
talkItem.setBeforeEvent("setDisableApproach", [true])
```

The next TalkItem makes Sam stand up again. It is also triggered by the Sentence “Use Barstool”.

This TalkItem is only executed when the GameEvent “SamAtBarstool” is set and it clears this GameEvent.

There is a position defined for this TalkItem to which Sam has to walk. So the user sees he is standing up.

```
// - leave barstool -
talkItem = sam.createTalkItem("Sam_LeaveBarstool", "Use_Barstool")

// only if not on barstool, clear game event
talkItem.setGameEvents("SamAtBarstool", null, null, "SamAtBarstool")

// show sam is rising from stool
talkItem.setStartPosition(455, 357)
talkItem.setStartDirection(0, 1)

// reactivate walking
talkItem.setBeforeEvent("setDisableApproach", [false])
```

We have a little problem with the **stacking** here. Because when Sam sits on the barstool, he is with his feet over the floor level. But the automatic stacking only works for objects at the same floor level. So Sam would appear behind the counter but not in front of the counter. We do a little trick here.

When Sam walks to the barstool, he has to pass the node "SamBarstool". When he enters this node, we disable the stacking, so his **z-index** is frozen. We do this by creating a TalkItem, which is triggered by the trigger event "enteringNode_SamBarstool". We use the before event to disable Sam's stacking.

```
// - suspend stacking -
talkItem = sam.createTalkItem(
     "Sam_EnteringNodeBarstool",
     "enteringNode_SamBarstool"
)

// freeze z-index
talkItem.setBeforeEvent("setStacking", [false])
```

Sam may have left the node, while sitting at the barstool. But when he leaves, he has to pass the node again. This is because, when the user clicks at a position and the controlled object is bound to a net, a path between two nodes of the net is calculated. The start node is the node next to the object. The target node is next to the mouse click. The walking object will pass the start node and stop at the target node.

So we will reactivate the stacking, when Sam is passing the node SamBarstool walking to one of the adjacent nodes SamFront or SamRight. Because we have two trigger events here, we provide an array of trigger events.

```
// - resume stacking -
talkItem = sam.createTalkItem(
     "Sam_LeavingNodeBarstool",
     [
           "leavingNode_SamBarstool_to_SamFront",
           "leavingNode_SamBarstool_to_SamRight"
     ]
)

// reactivate stacking
talkItem.setBeforeEvent("setStacking", [true])
```

8.5 Defining Default Talk Items

As you can see in Table 3 you can define default TalkItems for every command of the **SentenceController**. So when the user builds a sentence with the SentenceController to which is no action defined in the controlled object, the **default TalkItem** is executed. To make it more varied, we are defining more than one TalkItem for some of the trigger events. The actual response is picked randomly from this TalkItems.

```
// don't consume
sam.createTalkItem(
     "Sam_Consume_Default", "Consume_Default",
     "I think that's hard to digest."
)
sam.createTalkItem(
     "Sam_Consume_Default_2", "Consume_Default",
     "Its not my taste."
)
```

```
// don't use
sam.createTalkItem(
     "Sam_Use_Default", "Use_Default",
     "How should this work?"
)
sam.createTalkItem(
     "Sam_Use_Default_2", "Use_Default",
     "No."
)

// don't talk
sam.createTalkItem(
     "Sam_Talk_Default", "Talk_Default",
     "I think this can't talk."
)
sam.createTalkItem(
     "Sam_Talk_Default_2", "Talk_Default",
     "This will not hear me."
)

// doesn't look special
sam.createTalkItem(
     "Sam_Look_Default", "Look_Default",
     "I can't see anything special."
)
sam.createTalkItem(
     "Sam_Look_Default_2", "Look_Default",
     "Looks nice."
)

// don't give
sam.createTalkItem(
     "Sam_Give_Default", "Give_Default",
     "I can't give that."
)

// don't take
sam.createTalkItem(
     "Sam_Take_Default", "Take_Default",
     "I should not take it."
)

// don't close
sam.createTalkItem(
     "Sam_Close_Default", "Close_Default",
     "It doesn't look open."
)

// don't open
sam.createTalkItem(
     "Sam_Open_Default", "Open_Default",
     "I don't think I can open this."
)
```

You may put this default answers in a separate JavaScript file, to use them in each page, which uses the SentenceController.

8.6 Using Response Talk Items

We want, that the user have to make Sam sit on the barstool before he can order a drink. So we define two talk items which are triggered by the sentence “Talk Barkeeper” and executed regarding to our GameEvent “SamAtBarstool”.

When Sam isn't sitting on the barstool, he says “Hi!” and the barkeeper responds: “Hello young man.”

```
// - let sam say hello -
talkItem = sam.createTalkItem("Sam_Hello", "Talk_Barkeeper", "Hi!")

// while sam is not sitting
talkItem.setGameEvents(null, "SamAtBarstool")


// - barkeeper ansers hello -
talkItem = barkeeper.createTalkItem("Barkeeper_Hello", "Sam_Hello", "Hello
young man.")
```

When Sam is already sitting at the barstool, Sam says: “Please!” And the barkeeper responds: “What can I do for you?” We make this TalkItem clear the talk. We will see in a moment, what this is for.

```
// - sam ordering at barchair -
talkItem = sam.createTalkItem("Sam_Please", "Talk_Barkeeper", "Please!")

// while sam is sitting
talkItem.setGameEvents("SamAtBarstool", null)


// - barkeeper what can I do? -
talkItem = barkeeper.createTalkItem(
     "Barkeeper_WhatCanDo",
     "Sam_Please",
     "What can I do for you?"
)

// start new talk
talkItem.setClearTalk(true)
```

With a second TalkItem the barkeeper says nothing but presents the possible orders. These orders are added as **response TalkItems** to the talk item. If a TalkItem is executed, which has response TalkItems, the messages of the response TalkItems are presented to the user by the **TalkController**. So the user can select one of the response TalkItems to answer. The selected response TalkItem is executed like any other TalkItem. The object, to which the TalkController is assigned, executes it. This object is Sam.

We add four response TalkItems to the TalkItem of the barkeeper. The TalkItem of the barkeeper has four trigger events. So this “menu” of responses will be displayed again during the talk. The user can try one answer one after the other.

```
var responseTalkItem

// - barkeeper takes order -
talkItem = barkeeper.createTalkItem(
     "Barkeeper_Order",
     [
           "Barkeeper_WhatCanDo",
           "Barkeeper_John",
           "Barkeeper_Heidi",
           "Barkeeper_Any_Default"
     ]
)

// add orders as response talk items
talkItem.createResponseTalkItem(
     "Sam_Order_Beer", "May I have a beer please?"
)
talkItem.createResponseTalkItem(
     "Sam_Order_John", "Do you know where is John?", true
)
talkItem.createResponseTalkItem(
     "Sam_Order_Any", "Do you know somethig about %input%?"
)
responseTalkItem = talkItem.createResponseTalkItem(
     "Sam_Order_Nothing", "Nothing."
)
responseTalkItem.setHideMessage(true)
```

The TalkController displays the defined responses like this:

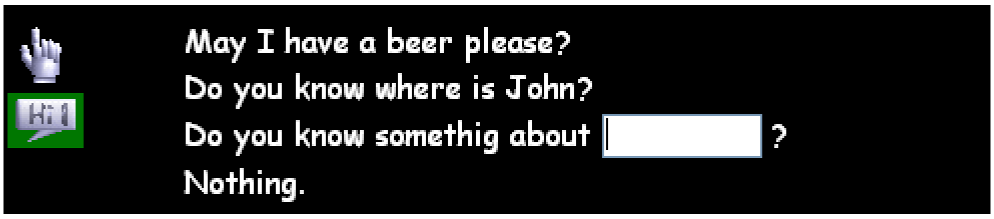

May I have a beer please?
Do you know where is John?
Do you know somethig about ?
Nothing.

Figure 16: Response Talk Items Displayed by the TalkController

Adding response TalkItems to a TalkItem is like adding TalkItems to an object. But there are no trigger events, because the display of the response TalkItems is triggered by the execution of the TalkItem, to which they are assigned.

```
var responseTalkItem = talkItem.createResponseTalkItem(responseTalkItemName,
message, onceAtTalk)
```

The method **createResponseTalkItem()** creates and adds the created response TalkItem to the TalkItem. It returns the new response TalkItem, which is just a TalkItem. So you can set additional properties to the response TalkItem.

We have assigned a response TalkItem in the above example to the variable responseTalkItem to set the property HideMessage to true. This property prevents Sam from speaking the message when the TalkItem is executed. When the user selects the response TalkItem “Nothing.” Sam says nothing, but the **TalkController** will be closed.

In the next sections we will respond to the other three response TalkItems. We will let ask Sam for John or any other person and we will let Sam order a beer.

8.7 Using a TalkItem only Once At Talk

In the previous section, we used **response TalkItems** to build something like a menu. We want this "menu" to be opened again during the talk, so the user can try out other questions.

When the user selects the second response TalkItem "Do you know where is John?" we want the barkeeper to answer and then display the menu again. But we want this menu item not to be displayed again, because it makes no sense to ask the same question over again.

We do this by setting the third parameter **onceAtTalk** of the response TalkItem to true. This makes this response TalkItem disappear until a TalkItem is executed, which has the property **clearTalk** set. So when the user starts the talk with the barkeeper again, this response TalkItem will be displayed again. So here is the code, which is repeating some code from the last section but is showing only the one response TalkItem for simplicity:

```
// - sam ordering at barchair -
talkItem = sam.createTalkItem("Sam_Please", "Talk_Barkeeper", "Please!")

// while sam is sitting
talkItem.setGameEvents("SamAtBarstool", null)


// - barkeeper what can I do? -
talkItem = barkeeper.createTalkItem(
     "Barkeeper_WhatCanDo",
     "Sam_Please",
     "What can I do for you?"
)

// start new talk
talkItem. setClearTalk(true)


// - barkeeper takes order -
talkItem = barkeeper.createTalkItem(
     "Barkeeper_Order", "Barkeeper_John"
)

// add orders as response talk items
talkItem.createResponseTalkItem(
      "Sam_Order_John", "Do you know where is John?", true
)

// add more response talk items here...


// - barkeeper responds about john -
talkItem = barkeeper.createTalkItem(
      "Barkeeper_John", "Sam_Order_John",
     "I haven't seen John for days."
)
```

So after the barkeeper has responded "I haven't seen John for days." the "menu" with the other response TalkItems is displayed again, because the TalkItem "Barkeeper_John" triggers it. But the question for John will not be shown again, until the TalkItem "Barkeeper_WhatCanDo" is executed, which clears the talk. Clearing the talk makes all TalkItems appear again, which where hidden by the property onceAtTalk.

8.8 Adding an Input Field to a Response Talk Item

You can add an **input** or **password field** to the message of a **response TalkItem**. This is done by including the placeholder **%input%** or **%password%** into the message. So the user has to enter a text in the input field and hit the Return key, to execute the response TalkItem.

You can use this functionality, to make the puzzles in your site more difficult for the user or to protect some behavior of your site by **password**. Or you can add something like a **help-functionality** to your site, which is searched by the entered keyword. Because the source code of your site is easily accessible to everyone, a password is no real security feature.

The difference between an input field and a password field is, that in a password field stars appear for the entered text. When the TalkItem with a password field is executed, the controlled object does not speak the message, because this would compromise the password.

To respond to a value of an input or password field, you have to provide one or more **keywords** instead of a trigger event for a TalkItem. We add a TalkItem, which responds to the keyword "Heidi". You may define more than one keyword for a TalkItem, by separating them by comma. Keywords are not case sensitive. When the user adds more than one word into the input field, the TalkItems are first searched for the entire text and then for each of the words to find a fitting TalkItem.

TalkItems with a fitting keyword are only searched in the object, which owns the TalkItem with the response TalkItem, which contains the input or password field. So in our case only the barkeeper can answer to the entered keyword.

This has the advantage, that we can add a **default TalkItem**, which is executed, if no TalkItem with a fitting keyword is found. We just have to use the name of the response TalkItem as trigger event. This default response talk item may use the same placeholder **%input%** to include the entered text in the response.

```
// barkeeper takes order
talkItem = barkeeper.createTalkItem(
     "Barkeeper_Order",
     [
           "Barkeeper_WhatCanDo",
           "Barkeeper_Heidi",
           "Barkeeper_Any_Default"
     ]
)

// add respond talk item with input field
talkItem.createResponseTalkItem(
      "Sam_Order_Any", "Do you know something about %input%?"
)

// barkeeper responds to input heidi
talkItem = barkeeper.createTalkItem(
     "Barkeeper_Heidi", null,
     "Heidi is not here today.", "heidi"
)

// barkeeper responds to any other input
talkItem = barkeeper.createTalkItem(
     "Barkeeper_Any_Default", "Sam_Order_Any",
     "I don't know anything about %input%."
)
```

The resulting conversation could be like this:

> Sam: *Do you know something about Ben?*
>
> Barkeeper: *I don't know anything about Ben.*
>
> Sam: *Do you know something about Heidi?*
>
> Barkeeper: *Heidi is not here today.*

8.9 Make Sam Order a Beer

In this section we will make Sam order a beer. The response TalkItem to init this order was added before.

Triggered by this response TalkItem, we let the barkeeper walk to the node at the beer tap "BarkeeperNodeTap". This command to walk is send by an event. "**walkToNode**". When the barkeeper reaches the node, we make him send an event "**hearEvent**" to himself. A call to the method **hearEvent(triggerEvent)** sends a trigger event to the barkeeper, which can be used to trigger the next TalkItem.

```
// - barkeeper brings beer -
// go to tap
talkItem = barkeeper.createTalkItem("Barkeeper_GoToTap", "Sam_Order_Beer")

// walk to tap
talkItem.setBeforeEvent(
     "walkToNode",
     [
           getNodeByName("BarkeeperNodeTap"),
           true, true,
           "hearEvent", ["Barkeeper_AtTapForBeer"]
     ]
)
```

We use this trigger event to make the barkeeper tap beer. We just let the barkeeper look south and wait for 4 seconds. To produce the waiting, we send the barkeeper the next trigger event "Barkeeper_TappedBeer" with a delay of 4000 milliseconds.

```
// - barkeeper taps beer -
// tap beer
talkItem = barkeeper.createTalkItem(
     "Barkeeper_TapBeer", "Barkeeper_AtTapForBeer"
)

// look to front
talkItem.setStartDirection(0, 1)

// post hear event
talkItem.setBeforeEvent("hearEvent", ["Barkeeper_TappedBeer"], null, 4000)
```

After the barkeeper has waited, we make him walk back to the node “BarkeeperNodeCenter” and chain the next trigger event “Barkeeper_BeerAtCenter”.

```
// - barkeeper walks back to center -
// go back to center
talkItem = barkeeper.createTalkItem("Barkeeper_BeerToCenter",
"Barkeeper_TappedBeer")

// walk to center
talkItem.setBeforeEvent(
     "walkToNode",
     [
           getNodeByName("BarkeeperNodeCenter"),
           true, true,
           "hearEvent", ["Barkeeper_BeerAtCenter"]
     ]
)
```

To put a beer on the counter, we have to create an object Beer. We add two looks to the object, which shows the beer in full and empty state. Then we make the beer invisible at the opening of the page.

```
// create beer
var beer = new MacaoObject("Beer", "Beer", 460, 148)

// add look Empty
beer.createLook("Empty", "optional/room/others/beerEmpty.gif", 48, 45)

// add look Full
beer.createLook("Full", "optional/room/others/beerFull.gif", 47, 48)

// make the full look active
beer.setActualLook("Full")

// set stacking just in front of the counter
beer.setZIndex(336)

// hide beer
beer.setVisibility(false)
```

When the barkeeper comes back to Sam, we switch the beer to the look Full and make it visible. To keep a marker about the state of the beer, we set a GameEvent “BeerFull”.

```
// - barkeeper put beer to counter -
// tap beer
talkItem = barkeeper.createTalkItem("Barkeeper_BringBeer",
"Barkeeper_BeerAtCenter")

// look to front
talkItem.setStartDirection(0, 1)

// change beer look
talkItem.setBeforeEvent("setActualLook", ["Full"], "Beer")

// afterwards the beer is full
talkItem.setGameEvents(null, null, "BeerFull", null)

// make beer visible
talkItem.setAfterEvent("setVisibility", [true], "Beer")
```

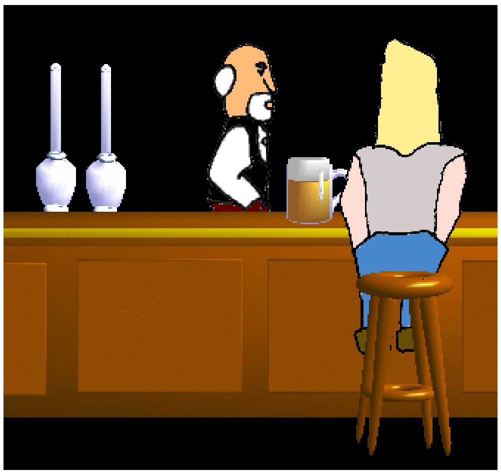


Figure 17: The Barkeeper Bringing the Beer

Our walk commands have cancelled the wandering around of the barkeeper. So we have to restart this.

```
// - barkeeper restart wander around -
talkItem = barkeeper.createTalkItem("Barkeeper_WanderAround",
"Barkeeper_BringBeer")

// start again wander around
talkItem.setBeforeEvent("wanderAround", [3500])
```

8.10 Making Sam Drink the Beer

Now we want, that the user can tell Sam to drink the beer.

To show Sam drinking the beer, we use the little gif-animation SamTake0.gif. This animation shows Sam from the back, which is direction 0, just moving the right arm a little. We add this animation as a Look to Sam.

```
// define look for taking something
sam.createLook("Take0", "optional/characters/sam/SamTake0.gif", 90, 230)
```

We want the user to build the sentence “Consume Beer” with the **SentenceController**, to make Sam drink the beer. So we use the trigger event “Consume_Beer”.

We hide the beer, let Sam say “Gurgle, gurgle, gurgle..” and display the beer again, now in the empty state. During this action we display the Look “Take0”, which we have added. This TalkItem can only be executed, when GameEvent “BeerFull” is set and it clears this GameEvent.

To have enough events, we split this in two TalkItems, where one triggers the other.

```
// - sam consume beer -
talkItem = sam.createTalkItem(
     "Sam_ConsumeBeer", "Consume_Beer",
     "Gurgle, gurgle, gurgle.."
)

// if beer is full. Afterwards the beer is empty
talkItem.setGameEvents("BeerFull", null, null, "BeerFull")

// make beer invisible
talkItem.setBeforeEvent("setVisibility", [false], "Beer")

// change beer look
talkItem.setAfterEvent("setActualLook", ["Empty"], "Beer")

// change Sam's look
talkItem.setLookName("Take0")


// - sam consume beer continues -
talkItem = sam.createTalkItem("Sam_ConsumeBeer2", "Sam_ConsumeBeer")

// make beer visible
talkItem.setBeforeEvent("setVisibility", [true], "Beer")
```

After this action an empty glas stands at the counter. If Sam orders another beer, the already defined TalkItems will cause the barkeeper to change the look of the beer to “Full” and set the GameEvent “BeerFull” again.

8.11 Let Sam Take the Empty Glass to the Inventory

Here we will see, how to use the **Inventory**. The Inventory is used to keep the items, which the controlled character takes around. The user can use the items of the Inventory to build sentences with the **SentenceController**. The items in the inventory stay available, when the character walks from page to page. The inventory also keeps its items, when you display another page in the inventory frame and open the inventory page later again. This works as long, as the **StorageManager** in the top frame stays alive.

We want to make Sam take the empty beer glass, when the user builds the sentence “Take Beer” with the SentenceController. So we use the trigger event “Take_Beer”.

This may only happen, after Sam has consumed his beer, so when the GameEvent “BeerFull” is cleared again.

When Sam takes the glass, we make the beer at the counter invisible and add the item “BeerGlass” to the Inventory. To add the inventory item, we send the event “**addInventoryItem**” to the object “Inventory” in the frameset frame “inventory”. We provide the internal name “BeerGlass”, the title “Beer Glass” and the URL of the image for the item.

The title of the item “Beer Glass” could be provided directly here. We call the method **getResource()** to assign the resource name “BeerGlassTitle” to this text resource and support later **internationalization**, which is not supported automatically here.

```
// - sam take glass -
talkItem = sam.createTalkItem("Sam_TakeBeerGlass", "Take_Beer")

// only when glass is empty
talkItem.setGameEvents(null, "BeerFull")
```

```
// hide the beer at the counter
talkItem.setBeforeEvent("setVisibility", [false], "Beer")

// put the glass to the inventory
talkItem.setAfterEvent(
      "addInventoryItem",
      [
            "BeerGlass",
            getResource("BeerGlassTitle", "Beer Glass"),
            "optional/room/others/beerEmpty.gif"
      ],
      "Inventory", "inventory"
)
```

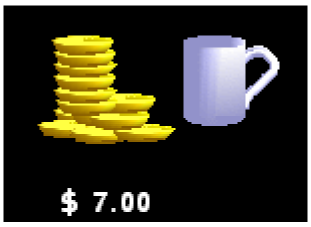


Figure 18: The Money and the Beer Glass in the Inventory

8.12 Let Sam Pay for the Beer

This section shows, how to work with items in the **Inventory**, which have an **amount**.

First we add an item “Money” to the Inventory. We provide the title “Money”, the image URL, the subtitle "<nobr>$ #.##</nobr>" and the actual amount, which is 7.

You can provide a **subtitle** to each inventory item. This subtitle contains a **format string** for the amount “#.##”. This format shows the amount with two decimal places. We have enclosed the subtitle in a nobr HTML-tag to make sure that there will be no line break in the subtitle.

We provide the target object name “Inventory” and the target **frame name** “inventory” for this event, because this event is fired when the content page is loaded. Providing the frame name ensures, that the event is buffered, until the Inventory page is fully loaded. When the frame name is not provided and the browser receives the inventory page after the content page, this event would be lost.

To make sure the money is only initialized once during a game, the initialization is guarded with a GameEvent MoneyInitialized. The initialization is only executed, if the GameEvent is not already set. After the initialization we set the GameEvent.

```
// the money is not initialized
if(! isGameEvent("MoneyInitialized")) {

      // put money to the inventory
      sendEvent(
            "addInventoryItem",
            [
                  "Money",
                  getResource("MoneyTitle", "Money"),
                  "optional/room/others/coins.gif",
                  getResource("MoneySubtitle", "<nobr>$ #.##</nobr>"),
                  7
            ],
            "Inventory", "inventory"
      )

      // set flag money initialized
      setGameEvent("MoneyInitialized")
}
```

To pay for one beer, we want the user to build the sentence “Give Money to Barkeeper”. So we use the trigger event “Give_Money_Barkeeper”.

We use three TalkItems to pay the beer.

The first TalkItem sends the event “**hasQuantity**” to the Inventory, to check if the inventory item “Money” has at least an amount of 3. The answer to this request is send as an event back to the object “Sam”. If the amount is larger or equal 3, Sam receives the trigger event “Sam_EnoughMoneyForBeer”. Else he receives the trigger event “Sam_NotEnoughMoneyForBeer”.

```
// - sam try to pay beer -
talkItem = sam.createTalkItem("Sam_TryPayBeer", "Give_Money_Barkeeper")

// check if there is enough money in the inventory
talkItem.setAfterEvent(
      "hasQuantity",
      [
            "Money", 3, "Sam",
            "hearEvent", ["Sam_EnoughMoneyForBeer"],
            "hearEvent", ["Sam_NotEnoughMoneyForBeer"]
      ],
      "Inventory", "inventory"
)
```

In the first case Sam says “Hey. Here you are for my beer.” And he sends an event to the Inventory to change the amount of the inventory item “Money” by the value -3.

In the other case Sam says “Ooops. I haven't enough money.”

```
// - sam pay beer -
talkItem = sam.createTalkItem(
      "Sam_PayBeer", "Sam_EnoughMoneyForBeer",
      "Hey. Here you are for my beer."
)

// reduce money in inventory
talkItem.setAfterEvent(
      "addQuantity", ["Money", -3],
      "Inventory", "inventory"
)


// - sam cant pay beer -
talkItem = sam.createTalkItem(
      "Sam_CantPayBeer", "Sam_NotEnoughMoneyForBeer",
      "Ooops. I haven't enough money."
)
```

If you want to execute more complex calculations or actions, you can add your own methods to the character. Call your method by a beforeEvent or afterEvent. Trigger TalkItems from your method by calling **hearEvent()** .

```
// add custom method to sam
sam.myMethod = function(param1, param2) {
      //...
      this.hearEvent("SomeTriggerEvent")
}

// call custom method by talk item
talkItem = sam.createTalkItem(...)
talkItem.setBeforeEvent("myMethod", [param1Value, param2Value])
```

8.13 Formatting Messages

The **messages**, which we are provide, when we are creating TalkItems or response TalkItems are HTML. So you can use HTML tags and **entities** to format the message. In addition line breads in a message are translated to <br> tags. The following table contains a few examples.

Effect	*Format*	*Example*
Line break	\n
	“Hello\nyoung man.” “Hello
young man.”
Quotation Marks	\” &quot;	“Hello \”young man\”.” “Hello &quot;young man&quot;.”
Italics	<i></i>	“Hello <i>young man</i>.”
Font Color	<font color='xxx'> </font>	"Hello <font color='red'>young man</font>."

Table 4: Formatting Messages

8.14 Changing the Bubble Style

This section shows, how to change the style of the **bubble**, using a **CSS style**.

We want to change the style of the bubble, which the Barkeeper uses to talk. So the user can better see, which text is said by Sam and which is said by the Barkeeper.

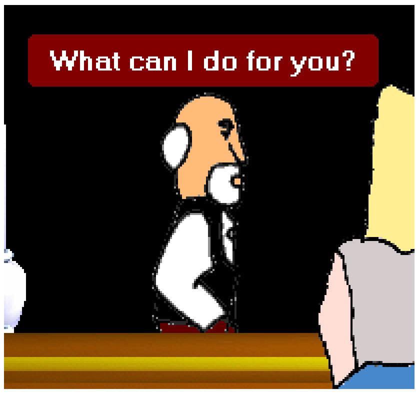


Figure 19: The Talking Barkeeper

We add a new Style section to the head of our HTML-page. In this section we build a style, which defines the new background color and the new font color for our bubble.

```
<style type="text/css">
.barkeeperBubble {
     background-color: darkred;
     color: white;
}
</style>
```

Note Don't name this style “barkeeper”, because this would change the style of the MacaoObject Barkeeper and the Barkeeper would get the new background color itself.

Now we assign the new bubble style to the barkeeper. We mix the original style class “bubble” with the new style class “barkeeperBubble”. Our new style class is the last, so it overrides the definitions of the style class “bubble”.

```
// change bubble style
barkeeper.setTalkCssClassName("bubble barkeeperBubble")
```

9 An Individual Frameset For Each Page

This chapter shows, how you can define an individual frameset for each content page. With individual framesets you can define different controller frames and controllers for each page.

You need individual framesets for each page, if you want to use persistence module to give the user the option to save the current game score. See chapter 10

The use of individual framesets is done by using nested frameset pages between the index page and the content pages.

9.1 Creating the Individual Framesets

First we need to create an individual frameset page for each content page. The frameset definition is pretty much the same as that, which we used for the index page. In the content frame, we set the link to the corresponding content page.

The frameset only needs to include the library **core/frameset.js**. If you forget to include this small library, commands to replace the frameset will have no effect.

Here you can see the source code of the file pubFrameset.html. This page needs no title tag, because the window title will be displayed from the index page.

```
<html>
<head>
     <meta http-equiv="expires" content="43200">
     <script language="JavaScript" src="../../core/frameset.js"
           type="text/javascript"></script>
</head>
<frameset
     rows="*,100"
     border=0 frameborder=0 framespacing=0
>
     <frame
           name="content"
           src="pub.html"
           marginheight=0 marginwidth=0
     >
     <frameset cols="50%,45%">
           <frame
                 name="controllers"
                 src="../../optional/controllers/controllers.html"
                 marginheight=0 marginwidth=0 scrolling=no
           >
           <frame
```

```
                  name="inventory"
                  src="../../optional/controllers/inventory.html"
                  marginheight=0 marginwidth=0
            >
      </frameset>
</frameset>
</html>
```

For the Road Map page we create a page roadMapFrameset.html. In this frameset we avoid the frame with the inventory. There are no actions at this page, for which the user needs the inventory items. Here is the source code:

```
<html>
<head>
<meta http-equiv="expires" content="43200">
<script language="JavaScript" src="../../core/frameset.js"
     type="text/javascript"></script>
</head>
<frameset
     rows="*,100"
     border=0 frameborder=0 framespacing=0
>
     <frame
           name="content"
           src="roadMap.html"
           marginheight=0 marginwidth=0
     >
     <frame
           name="controllers"
           src="../../optional/controllers/controllers.html"
           marginheight=0 marginwidth=0 scrolling=no
     >
</frameset>
</html>
```

9.2 Changing the Index Frameset

Now we reduce the frameset in the index page to a single frame. This frame we name **mainFrame**. The frame loads the roadMapFrameset.html at the start of the game.

```
<html>
<head>
<meta http-equiv="expires" content="43200">
<title>Tutorial Example</title>
<script language="JavaScript" src="../../core/storageManager.js"
type="text/javascript"></script>
</head>
<frameset
        rows="*"
        border=0 frameborder=0 framespacing=0
>
        <frame
                name="mainFrame"
                src="roadMapFrameset.html"
                marginheight=0 marginwidth=0
        >
</frameset>
```

```
</html>
```

Note You can add other frames beside the mainFrame for example to display a page with a logo in it.

9.3 Changing the Page Openings

At last we change the calls in the program, which are switching the pages. The calls now are opening the frameset pages into the mainFrame instead of opening the content pages into the content frame.

Here is the change for the roadMap.html:

```
// send an event to itself to open another page
talkItem.setAfterEvent("openPage", ["examples/tutorial/pubFrameset.html",
"mainFrame"], null)
```

Here is the change for the pub.html:

```
// open road map
talkItem.setAfterEvent("openPage", ["examples/tutorial/roadMapFrameset.html",
"mainFrame"], null)
```

Now you can enter the pub and take the beer glass into the inventory. When you leave the pub, the beer glass is not displayed any more, because the inventory is not displayed in the road map frameset. When you enter the pub again, the inventory with the beer glass is displayed again.

10 Saving and Loading the Game Score

This chapter explains how to use the module Persistence. With the use of this module the user can give the user the option to save and load the current state of the game.

Note Sometimes the Internet Explorer you may have a problem, when you try to open a page from the local file system providing an **URL parameter**. The persistence module uses an URL parameter when opening the Save Dialog and when restoring a game as a bookmark. An URL parameter is also used to start the game with another language.

To work around this problem copy Macao to a location, where the path contains no spaces and use the following form of the URL:

file:///C:/Macao/discover/index.html?openScore=WU0

Or use a Mozilla browser to test this functionality. How ever, this problem occurs only on the local file system. An application hosted in the Internet works fine.

10.1 How it Works

The browser has no access to the local file system to save a game score. And there is also no specific server required to host the game, which could be used to store the score. So the module Persistence provides a way for the user to save the game score as a bookmark in the browser. The user can maintain the list of saved games using the bookmark management of the browser.

Since the size of a link in a bookmark is limited, the game score is encoded. A Game Event for example is stored as a single bit in the score. To give the module Persistence the ability to encode and decode the game score, we need to register all items, which are to be included in the score. Don't worry, this will not be a lot.

The persistence module doesn't store the entire situation of the current page. It only stores the information, which is needed to restore the page like it was just entered by the user. What is actually stored is the following:

- The current page.
- The Opener Page of the current page.

- The **GameEvents**.
- The **inventory** items of one or more inventories.
- The used **language** of an internationalized game.
- Optionally other Storage Manager values.

What it doesn't save is for example the visibility, the position, the actual velocity and direction of the objects in the page. But you can use the Game Events and the Opener Page information to position the objects in a fitting way.

The next sections are explaining, how to integrate the module Persistence with our page.

10.2 Integrating the Save Dialog

First we need to load the package **core/persistence.js** into the index page after the package storageManager.js. When the application now is restarted, the save icon is displayed in the menu of the page controllers.html.

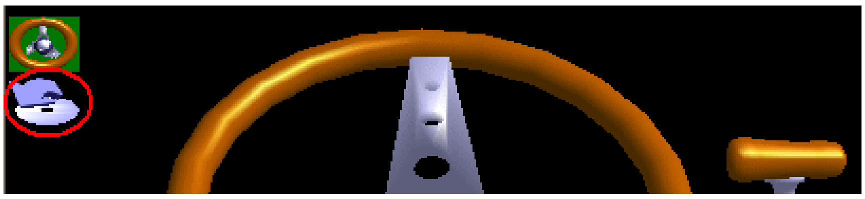


Figure 20: The Save Menu Item

Next we need to copy some files from the directory optional/controllers to the directory of our index file. Copy the following files:

- **save.html** (The **Save Dialog**)
- **save.gif** (An icon, which is displayed in the Save Dialog)
- **welcome.html** (The welcome page)

The page save.html contains the Save Dialog. This page is displayed to the user with the request to bookmark the page. The bookmark contains the link to the Save Dialog and the URL parameter **score**, which contains the game score. When the user opens the bookmark, the module Persistence in the dialog detects, that the Save Dialog was not opened by the game but by the user. So it opens the index page and forwards the game score to the index page.

Note The Save Dialog needed to be copied, because the position of the dialog defines, which application of a Macao installation is to be launched by the bookmark. A Macao installation may have several Save Dialogs for different games.

We need to adjust two references in the file save.html: First make sure the reference to the package persistence.js is right. Second make the parameter of the method **openStartPage()** reference the index page of the application. This reference is relative to the location of the Save Dialog and NOT relative to the base path.

```
<html>
<head>
<title>Save Game</title>
<style>
body {
     background-color: lightgrey;
}
body, td, input {
     font-family: Comic Sans MS, Arial, Helvetica, sans-serif;
     font-size: 12pt;
     font-weight: 590;
}
table {
     padding: 5px;
}

</style>
<script language="JavaScript"
     src="../../core/persistence.js"
     type="text/javascript">
</script>
<script language="JavaScript" type="text/javascript">

// start saved game if bookmark is reloaded
persistence.openStartPage("index.html")
...
```

Note If you want to change the text, which appears in the Save Dialog, you should not change the text in the file save.html. At runtime the internationalized text is retrieved from the file optional/controllers/controllers_resources.js and passed to the Save Dialog. Change the text in the resource file instead.

Now we need to do some changes in the index file.

For the use of the module Persistence the index file also needs to know the base path. Set the base path before loading the package storageManager.js.

```
// set base path
var basePath = "../../"
```

Create a script section for some code.

Including the package persistence.js into a file automatically creates a variable **persistence**. The variable references an object of the type **MacaoPersistence**. We will use this variable to perform the actions, which are needed to introduce persistence into the application.

We are calling the method **setSaveDialogUrl()** to show the index file how to open the Save Dialog.

```
// set save dialog url
persistence.setSaveDialogUrl("examples/tutorial/save.html")
```

We are letting the module Persistence start the application. For this purpose we change the target of the **mainFrame** to the page **welcome.html**, which we just copied. You can modify this page. But don't put too much afford in the styling of this page. The welcome page will only be visible as a background behind a dialog, while the user is asked, if a saved game is to be reloaded.

Now we need to tell the module Persistence to load the fitting page, dependent on if a saved game is to be loaded or a new game is to be started. For this purpose we are defining an event handler method onLoad(), which we are calling in the onload event of the frameset. In this method we call the method **startGame()**. For this method we need to provide the URL of the start page for the new game and a reference to the mainFrame, where the page is to be loaded.

For the use of the module Persistence we need to have individual framesets for each page (see chapter 9). So the entire page is referenced with one URL and all its controllers are reloaded.

```
..
// is called after load
function onLoad() {

     // start or restart game
     persistence.startGame(
           "examples/tutorial/roadMapFrameset.html",
           window.mainFrame,
           true
      )
}
</script>
</head>
<frameset
        onload="onLoad()"
        rows="*"
        border=0 frameborder=0 framespacing=0
>
        <frame
                name="mainFrame"
                src="welcome.html"
                marginheight=0 marginwidth=0
        >
</frameset>
</html>
```

10.3 Registering Score Elements

As mentioned before we need to tell the module Persistence which information is to be stored in the game score.

10.3.1 Registering Pages

We need to register all content pages of the game, which are actually two. Call the method **registerPage()** to register a page. The first parameter is the name of the content page file without the extension. These names must be unique. The second parameter is the URL of the frameset file relative to the base path. Here again you need an individual frameset for each content page. (See chapter 9)

```
// register pages
persistence.registerPage("roadMap", "examples/tutorial/roadMapFrameset.html")
persistence.registerPage("pub", "examples/tutorial/pubFrameset.html")
```

10.3.2 Registering Inventory Items

Next we register the **inventory items**. To register an inventory item call the method **registerInventoryItem()**. The first parameter is the name of inventory, which may contain the item. Since we are using the standard page inventory.html this is the name **Inventory**. The second and third parameters are the internal name and the visible title of the inventory item. The fourth parameter is the URL of the image, which is to be used to display the item.

In the game we have used the items Money and BeerGlass. The Money has also a quantity. So we need to provide three more parameters. The fifth parameter is the subtitle, which is used to display the quantity. The sixth parameter is the maximum quantity, which can be stored in the game score. The seventh parameter defines, if two decimal places are to be included in the score.

In a game score each element has its fix number of bits, which is used to save the information. So every score has the same size regardless of whether the user has a large or a small quantity of money. We need to provide the maximum value to reserve the space. If the actual value extends the reserved space, it is cut to the maximum value. The maximum value reserves the same space for positive and negative quantities. Because the persistence module internally calculates in bits, in this example the actual stored quantity of money can reach from -128.00 to 128.00.

```
// register inventory items
persistence.registerInventoryItem(
      "Inventory", "Money", "Money",
      "optional/room/others/coins.gif", "<nobr>$ #.##</nobr>",
      100, true
)
persistence.registerInventoryItem(
      "Inventory", "BeerGlass", "Beer Glass",
      "optional/room/others/beerEmpty.gif", null,
      0, false
)
```

Note Currently the quantity of menu items works with the data type **float**. That can cause rounding errors. This should be revised in a future version.

10.3.3 Registering GameEvents

Next we need to register the GameEvents. We have used only a few GameEvents.

You should not register GameEvents, which are only relevant during the visit of a single page. For example we have used the GameEvent SamAtBarstool, which is set while Sam is sitting on the barstool. This event will never be set, when Sam enters or leaves the page. But a score defines the situation, when the user enters the page. So we will not include this event into the score. If we included this event into the score, we also needed to place Sam on the barstool, if the page is opened and the event is set.

Reviewing the GameEvents in this manner, we find only one event, which needs to be stored in the score. This is the event MoneyInitialized. It makes sure, that the inventory item Money is only initialized once. Use the method **registerGameEvent()** to register the GameEvent.

```
// register game events
persistence.registerGameEvent("MoneyInitialized")
```

10.3.4 Registering StorageManager Values

Optionally you can include other values from the StorageManager into the score. To maintain these values we introduced the methods **setStorageValue()** and **getStorageValue()**. To register these values to the score use the methods **registerBooleanValue()**, **registerDecimalValue()**, **registerIntegerValue()** and **registerStringValue()**.

10.3.5 Registering Languages

If you build an internationalized application, you need to include the supported languages into the score. So the game will be restarted with the same language as played before. If your application only knows a single language, you don't need to register this language.

The tutorial example supports two languages. Use the method **registerLanguage()** to register the languages.

```
// register languages
persistence.registerLanguage("en")
persistence.registerLanguage("de")
```

Note The score registration can also contain some texts, which need to be internationalized. This especially concerns the inventory items and some messages of the module Persistence. You can use the method **openResourceExportWindow()** to export, translate and maintain the resource texts.

10.3.6 Registering the Version

Keep in mind, that information in the score is stored bit wise and there are no names to identify the information. So a score can only be loaded successfully, if the definition of the score, which consists of the registered elements, hasn't changed since the creation of the score.

To make sure an invalid score is rejected, you should maintain a version number with the score. If you register a version, the version is checked before the score is loaded. If the version number of the score doesn't fit, the loading is declined and the user has the only option to begin a new game.

It is recommended to start with a version number of 8. So four bits are reserved for the version number in the score. You should increase the version number each time before you publish a game with a changed score definition. Even the change of the ordering of registered elements like GameEvents means a change of the definition.

Use the method **registerVersion()**, to set the version number of the score definition.

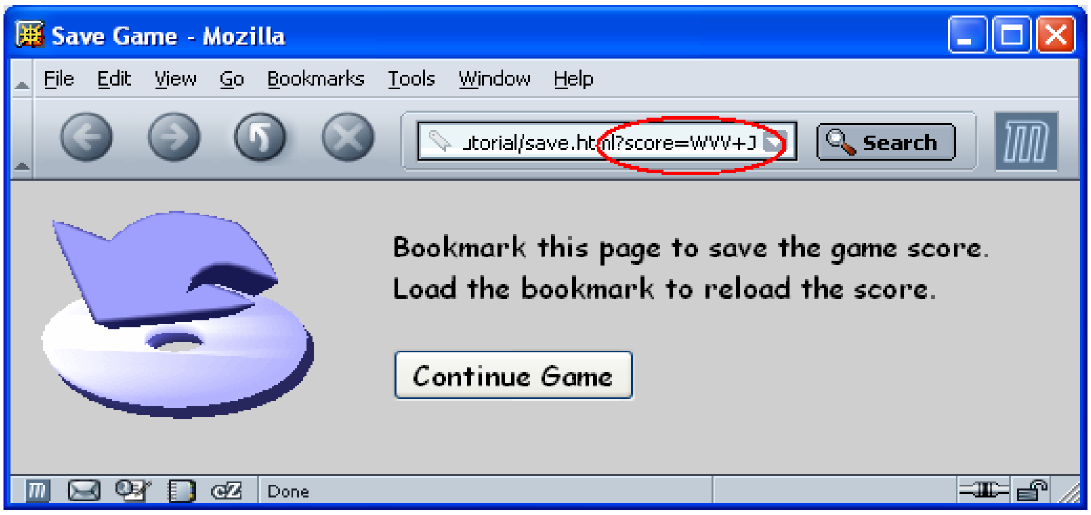
```
// register version
persistence.registerVersion(8)
```

10.4 Using the Persistence

Now that we have build in persistence to the application, let's test it. Start the application by opening the index file. You will notice, that the welcome page is not shown, because the start page immediately replaced it.

Go to the pub, order a beer, consume the beer, take the empty beer glass and pay the beer. Now click on the save icon to open the Save Dialog. The URL of the bookmark dialog contains the parameter score with the actual score. The score is Base 64 encoded. This means, 6 bits are stored as one character.

Bookmark the dialog in your browser and click "Continue Game" to close the dialog.

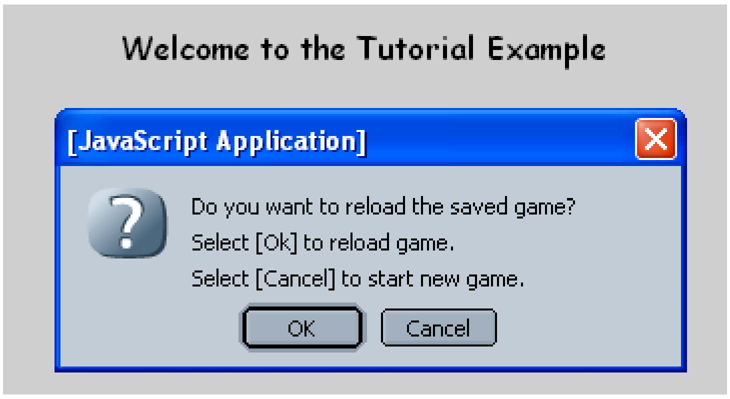

Save Game - Mozilla
File Edit View Go Bookmarks Tools Window Help
utorial/save.html?score=WVV+J
Search
Continue Game
Done

Bookmark this page to save the game score.
Load the bookmark to reload the score.

Figure 21: The Save Dialog

Now reload the just created bookmark. You will see a confirm dialog, which asks the user, if to restart the stored game or to begin a new one. If you want the score to be loaded without this question, set the parameter in the call of the method **startGame()** to false. (See page 78). In the background you can see the welcome page.

Click OK to load the game with the saved score. The pub page reopens. The beer glass is in the inventory and the money quantity is 4. Sam stands at the door.

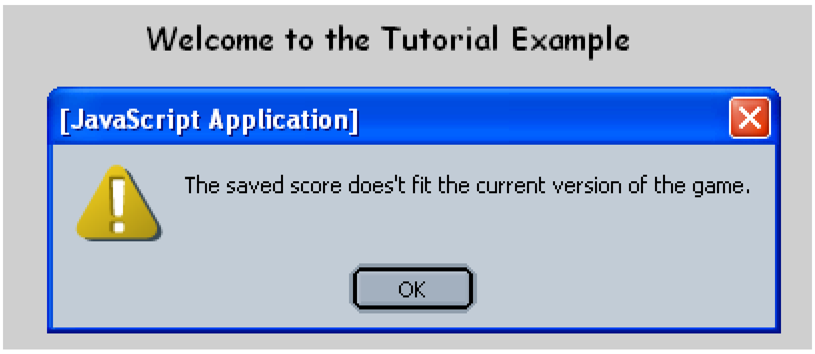

Welcome to the Tutorial Example

[JavaScript Application]

Do you want to reload the saved game?
Select [Ok] to reload game.
Select [Cancel] to start new game.

Figure 22: The Confirmation When Loading a Saved Game

If you change the registered version number of the score, you will get the following message, when reloading the saved game:


Welcome to the Tutorial Example

[JavaScript Application]

The saved score does't fit the current version of the game.

Figure 23: Message When The Score Version Changed

When the user clicks OK, a new game is started.

If you want to change these message texts, use the technique of the internationalization to export and redefine the text.

11 Creating a Card Game

In this chapter we will use the package **core/cards.js** to create a solitaire card game. You can find the code for this example in the folder examples/solitaire.

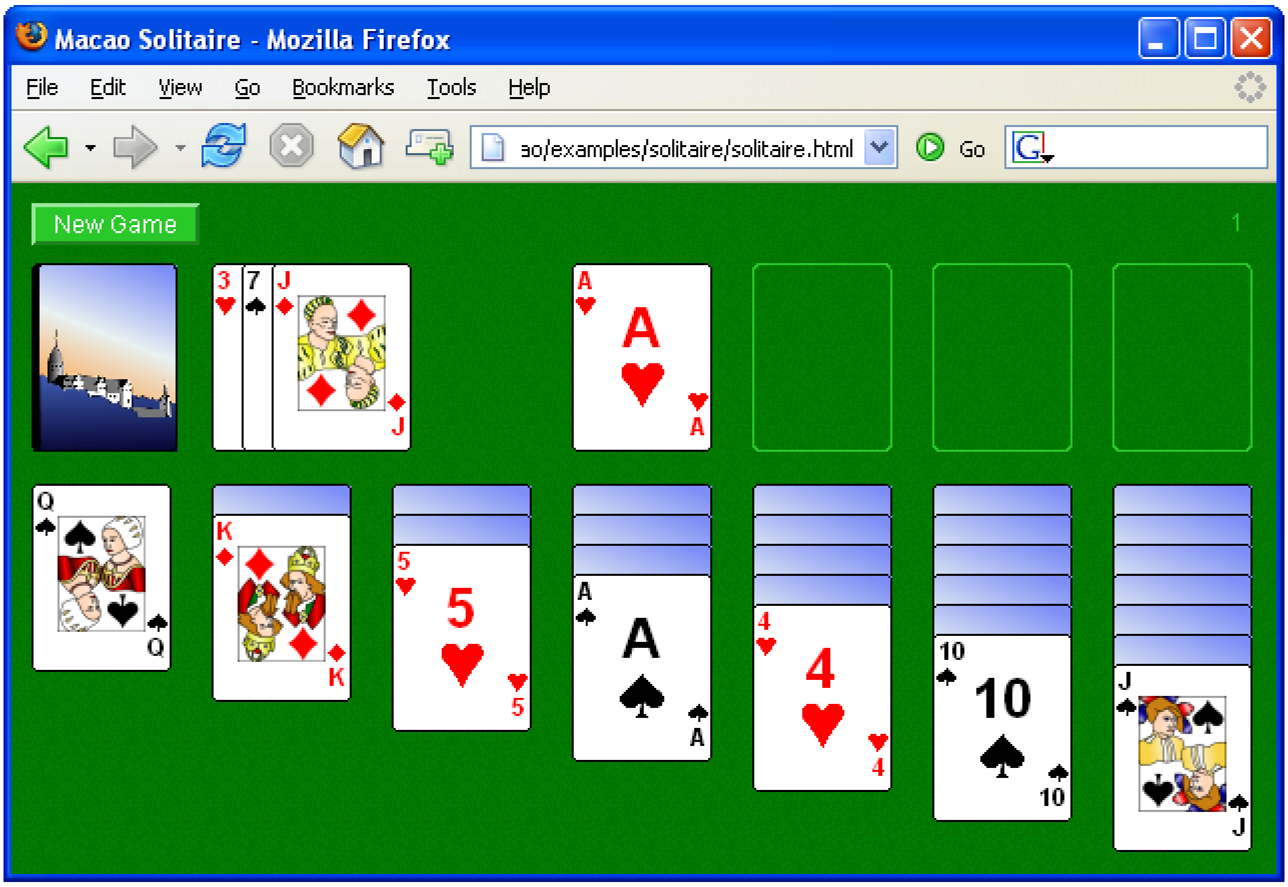

Macao Solitaire - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
ao/examples/solitaire/solitaire.html
Go
New Game
1

Figure 24: Macao Solitaire

11.1 Creating a Card Page

First we create an HTML page and add the required libraries. We don't need a frameset here. So we include the StorageManager in the page directly. We need the Kernel and the Dynamic library. Then the package **core/cards.js** is included.

For our game we use a standard set of cards, which is defined in the package **optional/cards/cardTypes.js**.

The package **optional/cards/cardTypeResources.js** defines the internationalized resource texts for the cards, which are currently English (“en”) and German (“de”). If you omit the resource package, you will only get English cards.

There are two stylesheets loaded. The stylesheet **core/macao.css** defines the style for Macao standard elements and is not really needed here. The example needs it for the win animation. The stylesheet **core/cards.css** defines the card layout.

Don't forget to set the **basePath**. This is the relative path between the page and the Macao directory.

```
<html>
<head>
     <title>Macao Solitaire</title>
     <meta http-equiv="expires" content="43200">
     <script>
           var basePath = "../../"
     </script>
     <link rel=stylesheet type="text/css" href="../../core/macao.css">
     <link rel=stylesheet type="text/css" href="../../core/cards.css">
</head>
<body
     style="background-color: green;"
     background="../../optional/cards/GreenCloth.png"
>
<script language="JavaScript" type="text/javascript"
     src="../../core/storageManager.js"></script>
<script language="JavaScript" type="text/javascript"
     src="../../core/kernel.js"></script>
<script language="JavaScript" type="text/javascript"
     src="../../core/dynamic.js"></script>
<script language="JavaScript" type="text/javascript"
     src="../../core/cards.js"></script>
<script language="JavaScript" type="text/javascript"
     src="../../optional/cards/cardTypeResources.js"></script>
<script language="JavaScript" type="text/javascript"
     src="../../optional/cards/cardTypes.js"></script>
<script language="JavaScript" type="text/javascript">

     // your code goes here

</script>
</body>
</html>
```

11.2 Creating a Standard Set of Cards

We use the method **createStandardCards()** from the package cardTypes.js to create a set of 52 cards without jokers.

The cards have ascending **card values** from 1 for the ace to 13 for the king. We will use these card values later for our game logic.

```
// create 52 cards
```

```
var allCards = createStandardCards(false, false, false, null)
```

11.3 Creating the Card Stacks

To organize the cards we use MacaoCardStack objects. A stack can be used to organize a deck as well as a hand of cards or a set of cards placed on the table.

Each stack is assigned an exposed schema and a drag schema. The exposed schema controls, which card of the stack is exposed, that means it shows its front side. The drag schema controls, which of the cards can be dragged using the mouse.

The cards on a stack are ordered from the bottom card to the top card.

You need to know the definition of a **substack**. As stubstack is defined as a list of cards of a stack beginning with a start card up to the top card. You can for example allow the user to drag a substack from one stack to another.

We create two stacks for the deck and the open deck in the top left corner of the page.

The deck gets the exposed schema None. So all cards on the deck will show their rear side. The drag schema None defines, that none of the cards can be dragged by the user. The cards have a y-offset of 0 and an x-offset of 0.2 pixels. The small x-offset lets the user estimate the number of cards of the stack.

```
// create deck
deck = new MacaoCardStack(
     "Deck", null, 10, 40,
     "Deck", true, 0, 0, 0.2, 0,
     MacaoCardStack.prototype.EXPOSED_SCHEMA_NONE,
     MacaoCardStack.prototype.DRAG_SCHEMA_NONE
)
```

Beside the deck we create an open deck for the open cards. On the open deck all cards are showing the front side (are exposed). The user can drag only the top card from the stack to drop it to another stack.

```
// create open deck
openDeck = new MacaoCardStack(
     "OpenDeck", null, 100, 40,
     "OpenDeck", true, 0, 0, 0.2, 0,
     MacaoCardStack.prototype.EXPOSED_SCHEMA_ALL,
     MacaoCardStack.prototype.DRAG_SCHEMA_TOP
)
```

Next we are creating seven play stacks in the second row. In this play stacks a substack of cards may be exposed. This substack can be dragged by the user to another stack. So the drag schema Exposed Substack is set.

A stack is a MacaoObject and needs a unique name. So every play stack gets another name from PlayStack0 to PlayStack6. To have a common identifier for the play stacks we assign each play stack the same **stack type** “PlayStack”.

The numeric parameters of the constructor are used to define the position of the stack and the offset between the stack and the first card and the offset between the cards.

```
var stackNo
var stack
var stacks = new Array()

// create seven stacks
for (stackNo = 0; stackNo < 7; stackNo++) {

      // create stack
      stack = new MacaoCardStack(
            "PlayStack" + stackNo, null,
            10 + (90 * stackNo), 150,
            "PlayStack", true, 0, 0, 0, 15,
            MacaoCardStack.prototype.EXPOSED_SCHEMA_SUBSTACK,
            MacaoCardStack.prototype.DRAG_SCHEMA_EXPOSED_SUBSTACK
      )

      // add stack to array
      stacks.push(stack)
}
```

At last we create four target stacks in the top right corner. All cards on these stacks are exposed. Only the top card can be dragged by the user.

```
var stackNo
var stack
var stacks = new Array()

// create seven stacks
for (stackNo = 0; stackNo < 4; stackNo++) {

      // create stack
      stack = new MacaoCardStack(
            "TargetStack" + stackNo, null,
            280 + (90 * stackNo), 40,
            "TargetStack", true, 0, 0, 0, 0.2,
            MacaoCardStack.prototype.EXPOSED_SCHEMA_ALL,
            MacaoCardStack.prototype.DRAG_SCHEMA_TOP
      )

      // add stack to array
      stacks.push(stack)
}
```

11.4 Shuffling the Cards an Dealing Out

Since we have created the cards and the stacks we can shuffle and deal out. We create a function to start this new game. We assign this function also to the button New Game, so the user can restart the game at any time.

We are broadcasting an event “removeAllCards” to all objects on the page. This causes the method **removeAllCards()** to be called at every stack. This removes the cards from the stacks, which may be there from the last game.

Then we call **getObject("Deck")** to get the deck.

We get a list of all cards from the global variable allCards, which we defined at the begin of this chapter. We add these cards to the deck by calling **addCards()**.

Here we need to pay attention on the performance. The method addCards() as well as many other methods for the manipulation of the cards on a stack has an additional boolean parameter **dontAdjustCards**. Setting this parameter to false causes the cards on the stack to be adjusted right after the operation. Adjusting the cards is a time consuming operation. It adjusts the **position**, the **stacking** (**z-Index**), the **exposure** and the **drag activation** of each card. So you should take care only to cause the adjustment once for each stack during your operation.

We set the parameter **dontAdjustCards** to true for each single operation and call **adjustCards()** in the end to adjust the deck only once.

After we have added the cards to the deck we shuffle the deck calling **shuffle()**.

Next we make all cards on the stack unexposed by calling the method **setAllExposed(false)** "manually". The adjustment of the cards would do this automatically. But we want to flip the cards before dealing out.

Then we iterate about the seven play stacks to deal out the cards. Every play stack gets another number of cards from the deck. A card can only be on one stack at a time. So calling **addCard(card)** on the play stack would implicitly remove the card from the deck. But calling first **removeCard()** with dontAdjustCards equals true at the deck prevents the deck from being adjusted for each card.

After all cards have been added to a play stack, the top card will be set exposed calling **setExposed()** and the cards on the stack are adjusted.

```
function startNewGame() {
      var deck
      var playStackNo
      var playStack
      var cardNo
      var card

      // remove cards from all stacks
      broadcastEvent("removeAllCards")

      // get deck
      deck = getObject("Deck")

      // add cards to deck and shuffle
      deck.addCards(allCards, true)
      deck.shuffle(true)

      // set cards unexposed
      deck.setAllExposed(false, true)

      // deal out to playstacks
      for (playStackNo = 0; playStackNo < 7; playStackNo++) {

            // get play stack
            playStack = getObject("PlayStack" + playStackNo)

            // process cards
            for (cardNo = 0; cardNo <= playStackNo; cardNo++) {

                  // get card from deck
```

```
                card = deck.getTopCard()
                deck.removeCard(card, true)

                // deal out card from deck
                playStack.addCard(card, true)
            }

            // expose top card
            card.setExposed(true, true)

            // adjust position and display
            playStack.adjustCards()
        }

        // adjust remaining cards on deck
        deck.adjustCards()
}
```

11.5 Getting Cards from the Deck

In the first part of this tutorial, which described an adventure game, we used **TalkItems** to control the behavior of the objects. In this part, we will instead **override methods** and add new methods to the classes. Changing a class will change every object, which is an instance of the class. So we only need to implement one method to control the behavior of all cards or all stacks.

When the user clicks on the deck, up to three cards are to be moved from the deck to the open deck. We build this functionality by implementing the event handler **onClickCard()** of the class **MacaoCard**. Because this event handler works for all cards, we first get the stack of the card and use the **stack type** to switch the logic.

```
MacaoCard.prototype.onClickCard = function(event, mouseX, mouseY) {
      var stack

      // get stack
      stack = this.getStack()
      if (stack) {

            // select stack type
            switch (stack.getStackType()) {

            case "Deck":

                  // the next code goes here
                  break

            case "PlayStack":

                  // the code for the play stacks goes here
                  break
            }
      }
}
```

We iterate up to three times. Each time we get the top card from the deck, remove it from the deck and add it to the open deck.

For the first of these three cards we set a reference named “fanStart” at the open deck. We will need this reference in a minute to build the fan of these three cards at the open deck.

At last we adjust the cards at the open deck. The deck doesn't need to be adjusted, because the remaining cards are staying unchanged.

```
var openDeck
var size
var substackIndex
var substackCard

// get open deck
openDeck = getObject("OpenDeck")

// process three cards
for (cardNo = 0; cardNo < 3; cardNo++) {

      // get top card from deck
      topCard = stack.getTopCard()
      if (topCard) {

            // keep reference to first card of fan
            if (cardNo == 0) {
                  openDeck.fanStart = topCard
            }

            // remove from deck without card adjust
            stack.removeCard(topCard, true)

            // move card to open deck, dont adjust stack
            openDeck.addCard(topCard, true)
      }
}

// adjust open deck
openDeck.adjustCards()
```

Now to the code for the play stacks. When a user clicks on the top card of a play stack and the top card is not exposed, it gets exposed. So we check, if the top card is clicked and if the card is not already exposed. Then we change the exposed state.

```
// is top card not excposed
if ((stack.getTopCard() == this) && !(this.isExposed())) {

      // expose top card
      this.setExposed(true)
}
```

When the deck is empty, the user can click on the empty deck. This moves all cards back from the open deck to the deck. For this functionality we implement the event handler **onClickStack()** of the class **MacaoCardStack**. Again we first remove all cards from the open deck. Then we use the method **reverse()** of the JavaScript class Array to revert the order in the array and maintain the order of the stack. Then we add the cards to the deck.

```
MacaoCardStack.prototype.onClickStack = function(event, mouseX, mouseY) {
      var openDeck
      var cards
```

```
     // select stack type
     switch (this.getStackType()) {

     case "Deck":

          // deck is empty
          if (this.isEmpty()) {

               // get the open deck stack
               openDeck = getObject("OpenDeck")

               // get all cards from open stack
               cards = openDeck.getAllCards()

               // remove cards from open stack
               openDeck.removeAllCards()

               // clear reference to first fan card
               openDeck.fanStart = null

               // revert ordering
               cards.reverse()

               // add cards to deck
               this.addCards(cards)
          }
          break
     }
}
```

11.6 Adjusting Cards on the Open Deck

When up to three cards from the deck arrive on the open deck, they need to build a fan. So the user can see that three cards have been drawn. To achieve this we implement the event handler **onAdjustCards()** of the class **MacaoCardStack**. This event handler is called after the standard adjustment of the cards has been executed. We can use this handler to override the standard positioning of the top cards.

Again we use the **stack type** to switch between the implementations. We will discuss the method refreshValueDisplay() for the target stacks later.

```
MacaoCardStack.prototype.onAdjustCards = function() {

     // select stack type
     switch (this.getStackType()) {

     case "OpenDeck":

          // the next code goes here
          break

     case "TargetStack":

          // refresh value display
          refreshValueDisplay()
          break
     }
}
```

We get the first card of the fan from the property fanStartCard, which we created before. Then we get all cards of the substack starting with this card, which will get up to three cards. If the fanStartCard is not element of the open deck any more, we will get an empty array.

We process these cards and move them to their fan positions. The third parameter of **moveTo()** is again dontAdjustCards. We must set this to true to prevent an endless recursive call, which would result in a stack overflow.

We need to be careful to not position a card, which is currently being dragged by the user or is “walking” to a target stack. We use the method **isPulling()** to skip such a card.

```
var fanStartCard
var fanCards
var fanCardNo
var fanCard

// adjust fan
fanStartCard = this.fanStart
if (fanStartCard) {

	// get fan substack cards
	fanCards = this.getSubstack(fanStartCard)

	// process fan cards
	for (fanCardNo = 0; fanCardNo < fanCards.length; fanCardNo++) {

		// get fan card
		fanCard = fanCards[fanCardNo]

		// if not card is being dragged or walked
		if (! fanCard.isPulling()) {

			// move x position
			fanCard.moveTo(
				this.getLeft() + (15 * fanCardNo),
				fanCard.getTop(),
				true
			)
		}
	}
}
```

11.7 Allowing Cards to be Dropped

When we created the stacks we defined which cards might be dragged by the user. Now we will control where these cards can be dropped. Drag and drop together defines the main logic of this game.

A card or a substack of cards can be dropped to a card or to a stack. Dropping a card to a stack means inserting it to the bottom of the stack. So we will only allow a card to be dropped to an empty stack. We override the method **isDropAllowed()** of the class **MacaoCardStack** to control, which card or substack may be dropped.

This method allows an object to be dropped but is not the drop operation itself. The method is called many times when a card or another object is dragged over the stack. If drop is allowed, the stack (or later the card) will get highlighted with a small border. The drop action is implemented by the event handler **MacaoCard.onDropToStack()**. This event handler moves the card or the substack of cards to the stack. You can override this event handler to get another behavior.

When you extend the game, there may be other objects, which could be dragged by the user. So we first test, if the dropped object is a card, by checking the attribute **IS_CARD**. Then we again use the **stack type** to distinguish between the stacks. We check, if the stack is empty.

Each card may be dropped to a play stack. Only an ace may be dropped to a target stack. Here we use the **card type** to distinguish between the cards.

```
MacaoCardStack.prototype.isDropAllowed = function(forObject) {
	var stackType
	var drop = false

	// is the dragged object a card?
	if (forObject.IS_CARD) {

		// select stack type
		switch (this.getStackType()) {

		case "PlayStack":

			// drop first card on empty stack
			drop = this.isEmpty()

			break

		case "TargetStack":

			// is the stack empty
			// and the dropped card is an ace
			drop = this.isEmpty() &&
				(forObject.getCardType() == "Ace")
			break
		}
	}

	return drop
}
```

A card or a substack of cards can also be dropped to another card. Dropping to a card means inserting the card or the substack into the stack just on top of the target card. So we only allow a drop operation to the top card of a stack. To allow this we override the method **isDropAllowed()** of the class **MacaoCard**. In this case the drop operation itself is executed by the method **MacaoCard.onDropToCard()**.

We use the **cardValue** to check, if the cards are dropped in the right order to the play stack or target stack. On a play stack a red suit and a black suit must alternate. We use the method **isRedSuit()** to check this. On a target stack all cards must have the same suit. We use the method **getSuit()** to check this.

```
MacaoCard.prototype.isDropAllowed = function(forObject) {
	var drop = false
	var stack

	// get stack
	stack = this.getStack()
	// is this card on a stack
	// and is this card the top card of the stack
	// and is the dropped object a card
	if (stack && (stack.getTopCard() == this) && forObject.IS_CARD) {

		// distinguish stack type
		switch (stack.getStackType()) {

		case "PlayStack":


			// is this card exposed and compare color and value
			drop = this.isExposed() &&
				(this.isRedSuit() != forObject.isRedSuit()) &&
				(
					this.getCardValue() ==
					(forObject.getCardValue() + 1)
				)
			break

		case "TargetStack":

			// is the dragged card a single card?
			if (forObject.getStack().getTopCard() == forObject) {

				// compare suit and value
				drop = (this.getSuit() == forObject.getSuit()) &&
					(
						this.getCardValue() ==
						(forObject.getCardValue() - 1)
					)
			}
			break
		}
	}

	return drop
}
```

11.8 Let a Card Walk to the Target Stack

Next we will implement a little convenience for the user. If the user double clicks on the top card of the play stack or open deck and the card fits to a target stack, it walks to the target stack.

To get this behavior we override the event handler **onDblClickCard()**.

```
MacaoCard.prototype.onDblClickCard = function(event, mouseX, mouseY) {
	var stack

	// get stack
	stack = this.getStack()
	if (stack) {

		switch (stack.getStackType()) {

		case "OpenDeck":
		case "PlayStack":

			// the next code goes here
```

```
                  break
            }
      }
}
```

We search through our array of target stacks to find a fitting stack. Then we use the methods isDropAllowed(), which we have just implemented, to check, if the card fits.

Then we use the method **walkToStack()** to let the card "walk" across the page to the target stack. This method works like MacaoCardStack.addCard(). It adds the card immediately to the top of the target stack. Only the position of the card "walks" across the page.

```
var stackNo
var stack
var stackName
var topCard

// is this the top exposed card of the stack
if (
      (this.getStack().getTopCard() == this) &&
      this.isExposed()
) {

      // process target stacks
      for (stackNo = 0; stackNo < targetStacks.length; stackNo++) {

            // get target stack
            stack = targetStacks[stackNo]

            // get target stack top card
            topCard = stack.getTopCard()

            // can the card be dropped on the empty stack
            // or on the top card of the stack?
            if (
                  stack.isDropAllowed(this) ||
                  (topCard && topCard.isDropAllowed(this))
            ) {

                  // move card to target stack
                  this.walkToStack(stack)

                  // end search
                  break
            }
      }
}
```

11.9 Summing up the Card Values

In this section we will use the cardValue property of the card to implement a game score.

We implement the method refreshValueDisplay(), which was mentioned above. This method iterates through the target stacks and builds the sum of the card values. The sum is set as HTML to a ValueDisplay object. This is a simple MacaoObject, whose creation is not shown here.

When the sum of 364 is reached, all cards are on the target stacks. The game has been solved by the user. We start a little win animation showing Sam. This uses a known technique. So its not again shown here.

```
function refreshValueDisplay() {
     var stackNo
     var stack
     var sum = 0

     // are the stacks already created?
     if (targetStacks) {

           // process target stacks
           for (stackNo = 0; stackNo < targetStacks.length; stackNo++) {

                 // get target stack
                 stack = targetStacks[stackNo]

                 // sum up value
                 sum += stack.sumCardValues()
           }

           // display value
           getObject("ValueDisplay").setHTML("" + sum)

           // is game won
           if ((sum == 364) && ! winAnimationStarted) {

                 // start win animation
                 startWinAnimation()
           }
     }
}
```

11.10 Creating a Custom Set of Cards

For the solitaire game we used a standard set of cards. This section shows you how to create your own cards.

There are two ways to create the look of a card. You can create it by filling out the elements of a generic form. This is called the generic way. Or you can provide a monolithic image for each side of a card. This is the image way. The standard set of cards is created using the generic way.

For the image way you need to create an image for the front side of each card. This is a lot of artwork. Loading this artwork to the browser slows down the start performance and produces a lot of web traffic. But you can style each card exactly the way you like it. And you can resize cards using the method **setZoom()** of the object. Use the methods **createFrontLookImage()** and **createRearLookImage()** to create front and rear look of a card using the image way.

Use the methods **createFrontLookGeneric()** and **createRearLookGeneric()** to create a card the generic way.

The front look of a generic card is divided into the five sections North, South, West, East and

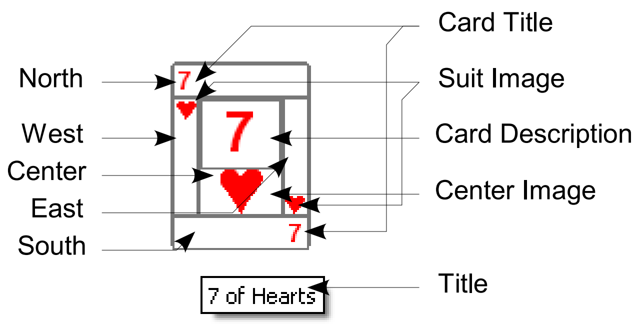


Figure 25: Card Sections

Center.

In the **North** and **South** the same **card title** is displayed.

In the **West** and **East** the same **suit image** is displayed. If you don't provide a suit image, there will be no West and East section created. So the Center is wider.

In the **Center** there can be a **center image** and / or a **card description**. The description can be displayed above or below the image. You can switch between a large and a small font for the description. The standard set of cards uses the large font, because the description consists only of one or two characters.

The **title**, which is provided with the constructor of the card, is displayed as tool tip. This can be a help to the user, when the cards are building a fan and are not fully visible. The tool tip is only shown, if the card is exposed. You can remove the tool tip by calling the method **setToolTip(null)**.

To create the “7 of Hearts” the following code is used:

```
// creating the card
card = new MacaoCard(
     "Hearts7", "7 of Hearts", 280, 10,
     "7", true, 0, MacaoCard.prototype.SUIT_HEARTS
)

// defining the front look
card.createFrontLookGeneric(
     "7", "red",
     "optional/cards/HeartsSmall.gif",
     "optional/cards/HeartsLarge.gif",
     "7", true, true
)

// defining the rear look
card.createRearLookGeneric(
     null, "optional/cards/RearCastleBlue.png", false
)
```

When you choose the small font, you can write an entire sentence to the card building event cards for example.

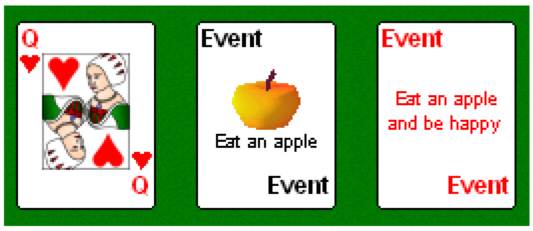


Figure 26: Custom Cards

You can find the source code for these cards under examples/cards/customCards.html.

If you want to resize the generic cards or you need other fonts, you should override the styles of the file **core/cards.css**.

12 Advanced Techniques

In this chapter we will see how to build a multilingual application and how to test the browser type before starting the application. You will get a few hints for putting your application to the Internet. And you will get advanced tips for JavaScript programming.

Other advanced techniques are described in the API documentation. Here is a list of topics:

Advanced Topic	***Where to find in the API Documentation***
Methods of walking	Package dynamic
How to use the StorageManager	Package storageManager
Creating your own menu	Class MacaoMenu
Creating your own controllers	Class MacaoMenu
Creating your own road elements	Class MacaoRoad
More trigger events for TalkItems	Class MacaoTalkItem
Many interesting methods	All Classes

Table 5: Advanced Techniques in the API Documentation

12.1 Internationalization

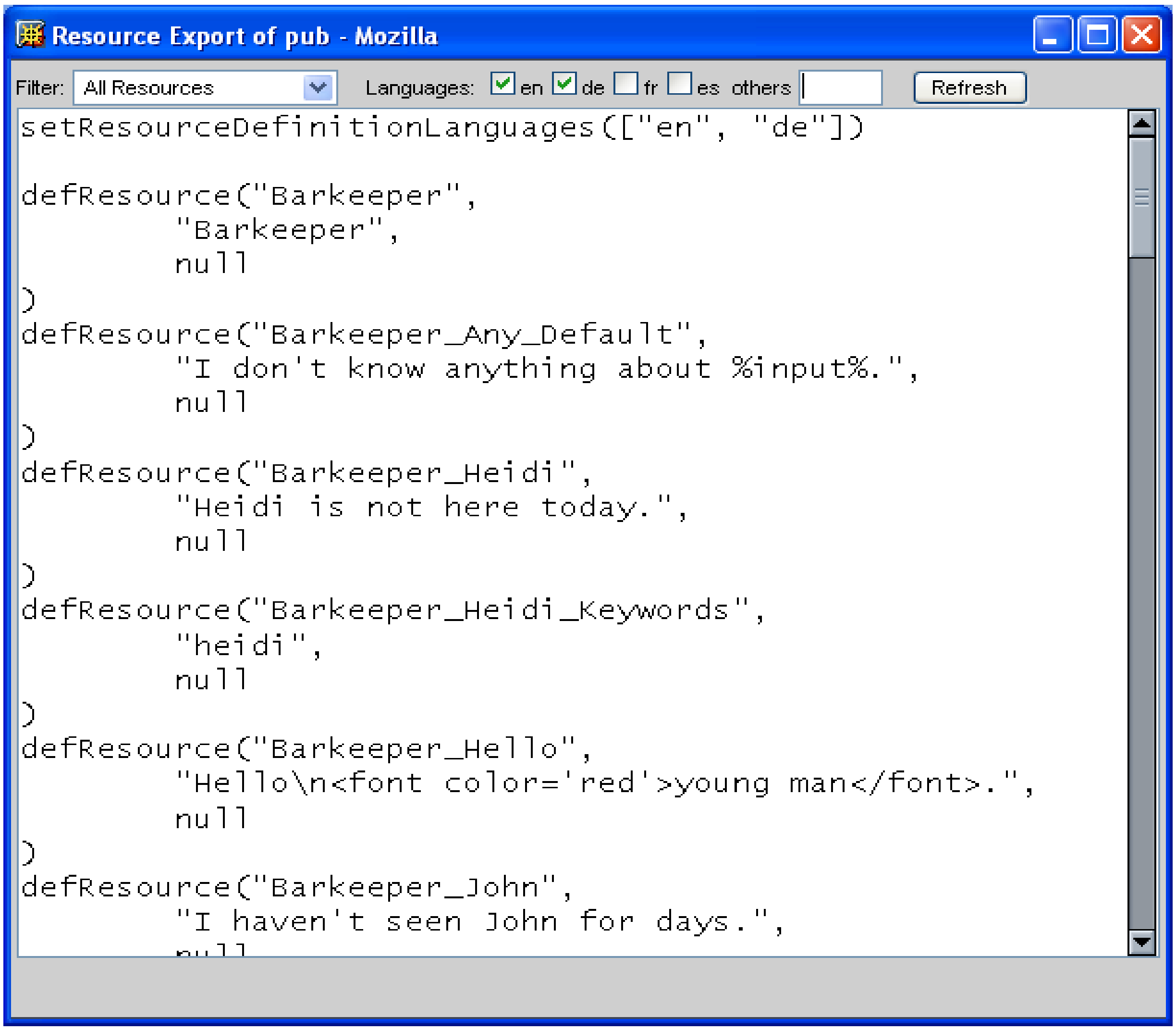
Macao has built-in **multilingual support**. You can use this to provide your pages dynamically in more than one language. If you are not interested in a multilingual application and have no need to maintain the text resources in an external file, you can skip this section.

All **messages** we have provided to the TalkItems and response TalkItems are automatically added to a pool of resources. In the pool a text resource is referenced by the name of the TalkItem. The titles of the objects are also added to the pool. They are referenced by the internal object names.

We can add additional resources to this pool by calling the method **getResource()**. We have used this method when adding items to the Inventory. The method takes the key and the text and returns the provided text. After we have done the internationalization it will return the text in the actual language.

```
getResource("BeerGlassTitle", "Beer Glass")
```

Each language is identified by a **language code**. The default language for new resources is “en” for English. So all resources of the TalkItems are added under the English language to the resource pool. You can change this definition language by calling **setResourceDefaultLanguage()**.


Resource Export of pub - Mozilla
Filter: All Resources
Languages: ✓ en ✓ de fr es others
Refresh

```
setResourceDefinitionLanguages(["en", "de"])

defResource("Barkeeper",
        "Barkeeper",
        null
)
defResource("Barkeeper_Any_Default",
        "I don't know anything about %input%.",
        null
)
defResource("Barkeeper_Heidi",
        "Heidi is not here today.",
        null
)
defResource("Barkeeper_Heidi_Keywords",
        "heidi",
        null
)
defResource("Barkeeper_Hello",
        "Hello\n<font color='red'>young man</font>.",
        null
)
defResource("Barkeeper_John",
        "I haven't seen John for days.",
```

Figure 27: The Resource Export Window

Now we use the **Resource Export** to export all resources to a separate JavaScript file. In this file we want to maintain the English and the German resources. To open the Resource Export window, we have to insert the method **openResourceExportWindow()** to the code of the page. Put this method after all other code to be sure that all resources are defined, when the export window is opened.

```
openResourceExportWindow()
```

The Resource Export window is opened in an extra window, so make sure to disable the **popup blocker**. In the window we check the checkbox “de” for the German language and click the button **Refresh**. In the textbox the JavaScript code is generated. The code defines the text resources for the languages English and German.

Select all text in the textbox and copy it to a new file, which we call pubResources.js.

Note You can select other additional languages by select a checkbox or entering the language code in the input field. You can enter more than one language code in the input field separating the codes by comma.

The first line of the generated code defines the languages of the following defResource() calls. The first resource parameter of all **defResource()** calls is filled with the English resources. The second resource parameter is still null. Here you have to enter the German translation in the JavaScript code.

```
setResourceDefinitionLanguages(["en", "de"])

defResource("Barkeeper",
     "Barkeeper",
     "Barkeeper"
)
defResource("Barkeeper_Any_Default",
     "I don't know anything about %input%.",
     "Ich weiss nichts über %input%."
)
defResource("Barkeeper_Heidi",
     "Heidi is not here today.",
     "Heidi ist heute nicht hier."
)
//..
```

Now we have to include our JavaScript file with the resource texts into the page. Add the link after the core/kernel.js but before the code, which defines the TalkItems. The resources of the JavaScript file will override the texts, which are set, when the TalkItems and objects are created. From now on you have to edit the resources in the new file.

```
...
<body style="background-color: black;">
<script language="JavaScript" src="../../core/kernel.js"
     type="text/javascript"></script>
<script language="JavaScript" src="pubResources.js"
     type="text/javascript"></script>
<script language="JavaScript" src="../../core/dynamic.js"
     type="text/javascript"></script>
...
```

Comment the call of the method **openResourceExportWindow()**. So the Export window is not opened any longer when the page is loaded.

Now when we start our little application again, it will still run in English, because the default language is English (en). But the resources come from the external file. To get the German version, you have to add the parameter language=de to the URL, when opening the index.html.

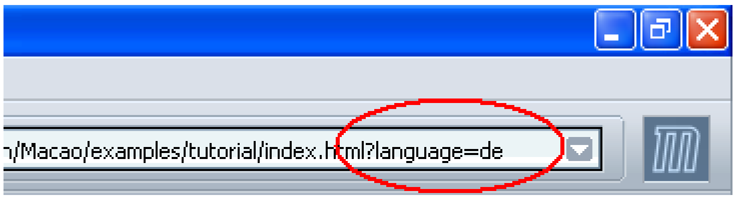


Figure 28: Language Parameter in the URL

If you open the Resource Export again, it will contain also the German texts. So when you add more resources to your page, you can repeat the export to generate the source for all resources.

Note Currently the resources for the controllers are only defined for the languages English (en) and German (de). If you want to use other languages, you need to edit the file optional/controllers/controllers_resources.js and enter the translations for the additional languages.

If you want to change the default language of the application, you can call **setLanguageSM()** in the page with the **StorageManager**, which is index.html.

```
<html>
<head>
<meta http-equiv="expires" content="43200">
<title>My Application</title>
<script language="JavaScript" src="../../core/storageManager.js"
     type="text/javascript">
</script>
<script language="JavaScript" type="text/javascript">
      setLanguageSM("de")
</script>
</head>
<frameset rows="*,100" border=0 frameborder=0 framespacing=0>
...
```

12.2 Checking the Browser Type

When you want to publish your Macao application to the Internet, you should start with a page, which checks the **browser type**. Because Macao runs on the most popular browsers but not on all browsers, there is the JavaScript package core/ **browserCheck.js**, which you can use to check the browser type.

Note The browser check is no guarantee of any kind that the application will run on the browser. And keep in mind that the browsers will change over time.

For the start page we create a small HTML page. The page includes the package browserCheck.js. It contains two links, one for the English and one for the German version of our application. Each of the links starts our function startApplication(), providing a **language code**.

```
<html>
<head>
```

```
<title>Start Tutorial Example</title>
<script
     language="JavaScript" src="../../core/browserCheck.js"
     type="text/javascript">
</script>
<script>

     // the code goes here

</script>
</head>
<body>
     <h1>Tutorial Example</h1>
     <a href="javascript:startApplication('en')">English</a>
     <a href="javascript:startApplication('de')">Deutsch</a>
</body>
</html>
```

Now we have to write the function startApplication(). It calls the function **isBrowserSupported()** from the package, to check the browser type.

If the browser check succeeds, we open the frameset page of our application, providing the selected language code as a parameter. The parameter language is optional. If you omit this parameter the application will start with the default language English (en).

If the browser check fails, we call **displaySupportedBrwosers(language)** to display a messagebox, which lists the supported browsers. This method currently only knows the language “de” for German. All other parameters will result in the display of an English message.

```
function startApplication(language) {

     if(isBrowserSupported()) {

           // start application
           window.location = "index.html?language=" + language
     } else {

           // display error message
           displaySupportedBrwosers(language)
     }
}
```

12.3 Adding a Splash Screen

When your start page needs a little longer to load over the Internet, you might want to display a **splash screen**, which is displayed during the loading of the rest of the page. In addition to the loading of the files the rendering of the road grid may take some seconds.

You should first create an image for the splash screen. This image may have transparent areas.

Use the method **showPopup()** just after the loading of the kernel.js to show the image as splash screen.

The popup uses the **Bubble** object, which is used to display the speech text elsewhere in the page. But we don't want to show the splash screen in the yellow bubble style. So we define a new CSS style “transparent” with no properties and provide it with the method showPopup(). This replaces the bubble style.

We send the bubble an event to set the z-index to a large number. 3000 is greater than the height of the page in pixels. So the bubble will be displayed in front of other objects. The z-index of the most other objects is set by the stacking mechanism to the value of the y-position of their bottom edge.

We use the method **getOpenerPageName()** to check, that the page is not opened by another Macao page. So the splash screen is only shown, when the application is started.

We use the method **hidePopup()** to hide the splash screen, after the rest of the JavaScript has been loaded and initialized.

```
<head>
<meta http-equiv="expires" content="10000">
<script>
     var basePath = "../../"
</script>
<style>
.transparent {
}
</style>
<link rel=stylesheet type="text/css" href="../../core/macao.css">
</head>
<body>
<script language="JavaScript" src="../../core/kernel.js"
type="text/javascript"></script>
<script>

// check is application started
if(! getOpenerPageName()) {

     // show splash screen
     showPopup(
           null, null,
           "discover/DiscoverLogo.gif", 476, 202,
           "transparent", null
     )

     // set popup to front
     sendEvent("setZIndex", [3000], "Bubble")
}
</script>
<script language="JavaScript" src="../../core/dynamic.js"
type="text/javascript"></script>
<script language="JavaScript" src="../../core/road.js"
type="text/javascript"></script>
<script language="JavaScript" src="../../optional/road/roadTypes.js"
type="text/javascript"></script>
<script language="JavaScrsipt">

//... the page code goes here ...

// close popup
hidePopup()
</script>
</body>
```

Note This sample code is taken from the page city.html of Discover Macao.

12.4 Setting the Walk Interval

This chapter is about the fine-tuning of your application. By setting the **walk interval** you can control the step duration for all walking objects (characters, cars, cards etc.) on a page. Call the method **setWalkInterval(duration)** to change the walk interval. Provide the duration in milliseconds. The default value is 150 milliseconds.

12.4.1 How it Works

The walk interval is the desired waiting time for the objects between two steps. But it is not the entire step duration. The **step duration** is the sum of the **calculation duration** and the **walk interval**.

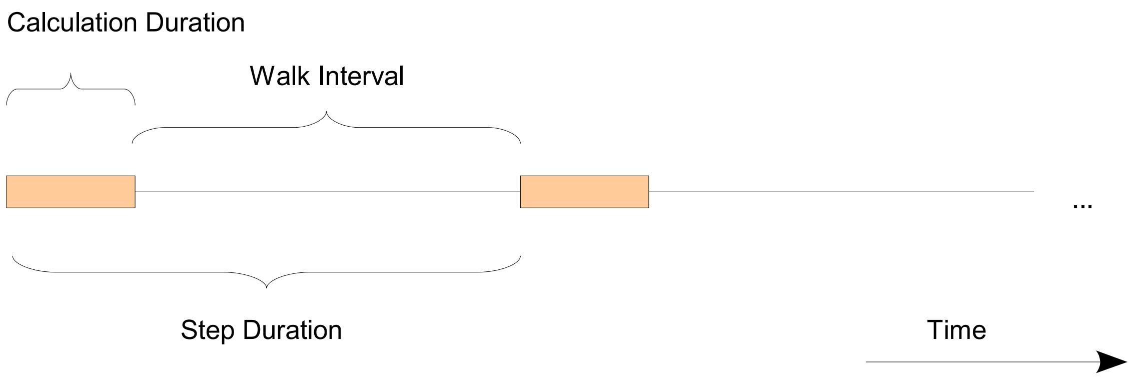


Figure 29: The Walk Interval

The calculation duration is the time needed by the application to calculate the next position of the object and the time of the browser to repaint the object.

Note The Opera browser is doing the repaint asynchronously. So intermediate results of a calculation can become visible.

Reducing the walk interval lets the objects do more steps per second and so makes the steps look smoother. But that's not the entire truth.

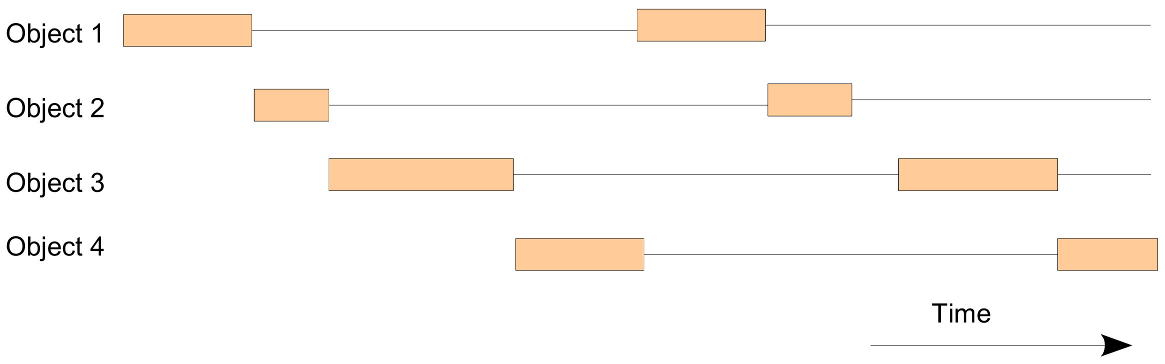


Figure 30: Multiple Walking Objects

If you have more than one object walking at the same time, each of the objects needs its calculation duration. The browser executes the script calculations sequentially. If you choose the walk interval too small, the browser will automatically extend it. The browser will calculate the other objects first, before the next step is started. This will result in unsteady speed dependent on how many objects are walking at a time.

If you choose the walk interval too small in comparison with the calculation duration, the speed of the objects will be highly dependent on the performance of users computer and its graphics card. Keep also in mind, that the computers will get faster over time.

If you animate a walking character, each walking phase will show another look. So you need a step duration, which fits your animation. If it runs too fast, it might look funny. This is no problem for objects like cars. These do not change the look on each step.

If you change the walk interval, the walk parameters like **velocity**, **maxAngularVelocity** and **curveBreak** are scaled before they are used for the calculation of a step. If you for example reduce the walk interval to 75 milliseconds, which is the half of the default value, the parameters will be multiplied with the factor of 50%. If the object has a velocity of 20, it will only walk 10 pixels per step. This helps you experiment with the walk interval without the direct need to adjust the walk characteristics of the objects.

This scaling works fine as long as the walk interval is large compared with the calculation duration. Setting the walk interval to 0 will all objects cause to stay at their position, even if the steps are executed. You can use the method **MacaoObject.setWalkCharacteristics()** to fine tune the walk parameters of the objects.

```
// method signature
MacaoObject.setWalkCharacteristics(
     maxVelocity, maxAcceleration, maxAngularVelocity, curveBreak
)
```

Note Setting a zoom factor to an object also scales some walk characteristics for their use. An object with a zoom factor of 50% will only walk with 50% effective velocity.

12.4.2 Recommendations for the Walk Interval

These are enough dizzy internals. Here are a few recommendations on how to set the walk interval:

- The default value should do good work for all pages with walking characters. Reduce it only, if you provide more animation phases for each step of a character.
- Reduce the value for a page with a road map to 100 or 75 milliseconds. This makes the cars move smoother.
- Test the result on computers with different performance.
- Over the years of improved computer performance you may check, if it makes sense to reduce the value for pages with road maps.

12.5 Deactivating the Design Mode

By default the **Design Mode** is active. When the Design Mode is active, the user can do some actions, which are useful during design time:

- The hit of the **key E** opens the Road Editor, if a road page is focussed.

- Holding the **Shift** and the **Control key** down, the user can drag objects with the mouse. You can use this feature to find an adequate position for the objects.
- Holding this keys down, the position of the mouse is displayed in the status bar.

To deactivate the Design Mode call the method **setDesignModeSM(false)** in the page with the Storage Manager.

```
// deactivate the design mode
setDesignModeSM(false)
```

12.6 Publishing an Application

When you publish your page to the Internet, you should keep some things in mind.

When you want to use free **hosting** offerings, the hoster will normally add some HTML and JavaScript to your pages for advertising. This advertising may cover some of your page or might hinder the use of your application. So may be you should use a commercial hosting.

Because a page normally contains a lot of images and source code, you should watch the **traffic** of your homepage. Many hosting offerings contain a fixed transfer volume. Exceeding this limit will cause additional cost.

Many larger companies use **proxies** with tools like Web Washer, which may add JavaScript to incoming HTML pages. This JavaScript disables some dynamic functions of the browser. The users in the net of these companies will not be able to execute the application. Normally the animation will not run.

During runtime Macao will only open the Save Dialog as a new window. If a **popup blocker** is activated, this should be no problem. Most popup blockers measure the time between the last mouse click of the user and the time when the window requested to be opened. If the time is short enough, the popup blocker assumes, that the mouse click requested the opening of the window and will allow it. If the opening is prevented, Macao displays a message box with a hint.

Because the application is build of **JavaScript**, the user needs to activate or accept JavaScript. If the Macao application is run locally, the Internet Explorer blocks JavaScript and displays a message bar with a warning in the browser. The user needs to allow the active scripting. This blocking doesn't happen, if the application is started from the Web. If you want to make the Macao application run in the local file system without the warning, you might consider making an **HTML application** of it. Just change the extension of the index.html to **.hta**. See http://msdn.microsoft.com/library for more information about HTML applications.

12.7 Advanced JavaScript Tips

12.7.1 Extending Objects

When you create an object using a class, you can extend it with your own **methods**. The following example adds the method sayHello() to the object sam and calls it.

```
// create an object
var sam = new MacaoObject("Sam", "Sam", 54, 224)

// add a method
sam.sayHello = function(toName) {

     // the code of the method
     alert("Hello " + toName)
}

// call the method
sam.sayHello("Frank")
```

You can also add **properties** to the object. You create a property by assigning a value to it. If you are in a method, you have to use the keyword **this** to access properties and other methods of an object.

```
// setting a property directly
sam.friendName = "Frank"

// or via method
sam.setFriendName = function(name) {
     this.friendName = name
}

// setting the name
sam.setFriendName("John")
```

12.7.2 Overriding Methods

When you are an advanced user of the Macao API, you might want to override some of the standard behavior of the Macao classes for your own objects. You can do this just by adding a method with the same name to your own object.

Sometime you may want to call the method of the super class in your overriding method. In the prototype-based language JavaScript, there is no keyword “super” to access the super class. But you can assign the method from the super class to your object using another name. Then you can call it in your overriding method using the other name.

```
// keep the super method
sam.superCalculateStackingIndex = sam.calculateStackingIndex

// override the method calculateStackingIndex
sam.calculateStackingIndex = function calculateStackingIndex() {

     // use the super method and add some value
     return 10 + this.superCalculateStackingIndex()
}
```

13 References

In this chapter you get some links to useful external resources.

13.1 Language References

13.1.1 DHTML Reference of Microsoft Internet Explorer

To get the DHTML object-reference for the Internet Explorer open the page http://msdn.microsoft.com/library and navigate to the menu item *Web Development – HTML and Dynamic HTML – SDK-Documentation – Reference*.

Under *Web Development – Cascading Style Sheets* you can find a CSS reference.

Under *Web Development – Scripting – Documentation – Windows Script Technologies – JScript – Reference* you can find a reference of the language JScript. JScript is the Microsoft implementation of Java Script.

Under *Web Development – HTML-Applications* you can find a reference on how to create stand alone HTML applications for Windows.

Under *Web Development – Scripting – Documentation – Windows Script Technologies – Script Runtime – FileSystemObject Object* you can find a reference of the **FileSystemObject**. You can use the FileSystemObject to access the local file system from stand-alone applications.

13.1.2 DHTML Reference of Mozilla

Under http://www.mozilla.org/docs/dom/domref/ you can find the reference of the Mozilla DHTML objects (it's hard to find). You can download this as zipped HTML under
http://www.mozilla.org/docs/dom/domref.zip or as PDF under
http://www.mozilla.org/docs/dom/domref.pdf.

13.1.3 DHTML Reference of Opera

For the functions that are supported in Opera 7 see http://www.opera.com/docs/specs/.

13.1.4 SELFHTML

SELFHTML is a cross browser reference for HTML, DHTML, CSS, JavaScript and more. While originally created in German, it is translated to France, Spanish, Japanese and English. You can find it at http://www.selfhtml.org/.

13.1.5 W3Schools

On the website http://www.w3schools.com you can find tutorials for HTML, DHTML, CSS and Java Script.

13.2 Tools

Below you will find resources with tools. You may use the tools to edit the HTML, the Java Script and the images of your application.

13.2.1 The Gimp

The Gimp is the most famous open source graphic editor for Linux and Windows. You may download it from http://www.gimp.org.

13.2.2 ImageMagick

ImageMagick is an API and a set of command line programs, which are used to modify images. You need to install ImageMagick in order to use the Macao tool **Rotate**. You can download ImageMagick from http://www.imagemagick.org.

13.2.3 Inkscape

Inkscape is a very good open source vector graphics editor. You can download it from http://www.inkscape.org.

13.2.4 JDraw

JDraw is a nice little pixel editor for GIF and PNG images. You can use it to create GIF-animations. You can download it from http://jdraw.sourceforge.net. You have also to download the fitting Java runtime environment from http://java.sun.com.

Starting it is a little tricky. I used a .bat file in the JDraw-directory with the line:

```
C:\j2sdk1.4.1_02\bin\java -jar jdraw_jdk1.4.jar Jdraw
```

13.2.5 jEdit

JEdit is an open source editor written in Java. You can use it to edit HTML and Java Script. It has many plugins for several purposes. You can download jEdit from http://www.jedit.org. The plugins are downloaded via the Plugin Manager of the editor.

13.2.6 Notepad++

Notepad++ is another open source editor for several programming languages. You can find it at http://notepad-plus.sourceforge.net/uk/site.htm

13.2.7 Open Office Draw

The drawing editor of Open Office is a nice tool to create images of simple 3D objects. See http://www.openoffice.org/

13.2.8 png2ico

Png2ico is a nice little tool, which you can use to create a **favicon** for your homepage or an icon for your Windows programm. You can find the tool at http://www.winterdrache.de/freeware/png2ico/index.html

13.2.9 Proton

Proton is a programmer's text editor for Windows. It's available in English and German language. You can download it from http://www.meybohm.de.

13.2.10 WTP

The **Eclipse Web Tools Platform Project** contains several tools for development including JavaScript development. It might be a little over sized for JavaScript development and it's not easy to install, even if you can use the Eclipse update manager. But it's the only open source editor for object oriented JavaScript programming that I know. By the way, I use my own self made editor. See http://www.eclipse.org/webtools/

14 Alphabetical Index