Managing Asynchronicity

Douglas Crockford

More than one thing at a time.

So hard.

The Law of Turns

- Never block.
- Never wait.
- Finish fast.

The Problems With Threads

- Races
- Deadlocks
- Reliability
- Performance

- Threads are safest and fastest when they don’t share memory.

Event driven systems

- Turn based. No pre-emption.
- Associate events with actions.
- Easy (beginners can do it).
- User interfaces.


Archivio Composizione Esposizione Strumenti
The Macintosh™ Finder
Bruce Horn and Steve Capps
512K
Version 4.1 ©1985 Apple Computer

File Edit Go Tools Objects

Home Card

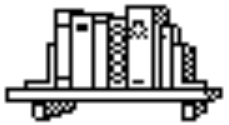

Intro


Help


Address

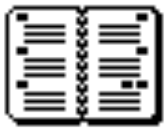

Documents

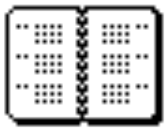

File Index


Book Shelf

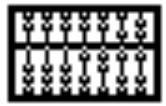

Phone


To Do


Weekly


Calendar


Slide Show


HyperCalc


Art Ideas

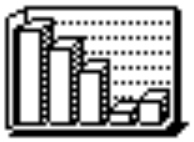

Clip Art


Card Ideas

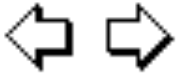

Button Ideas


Stack Ideas

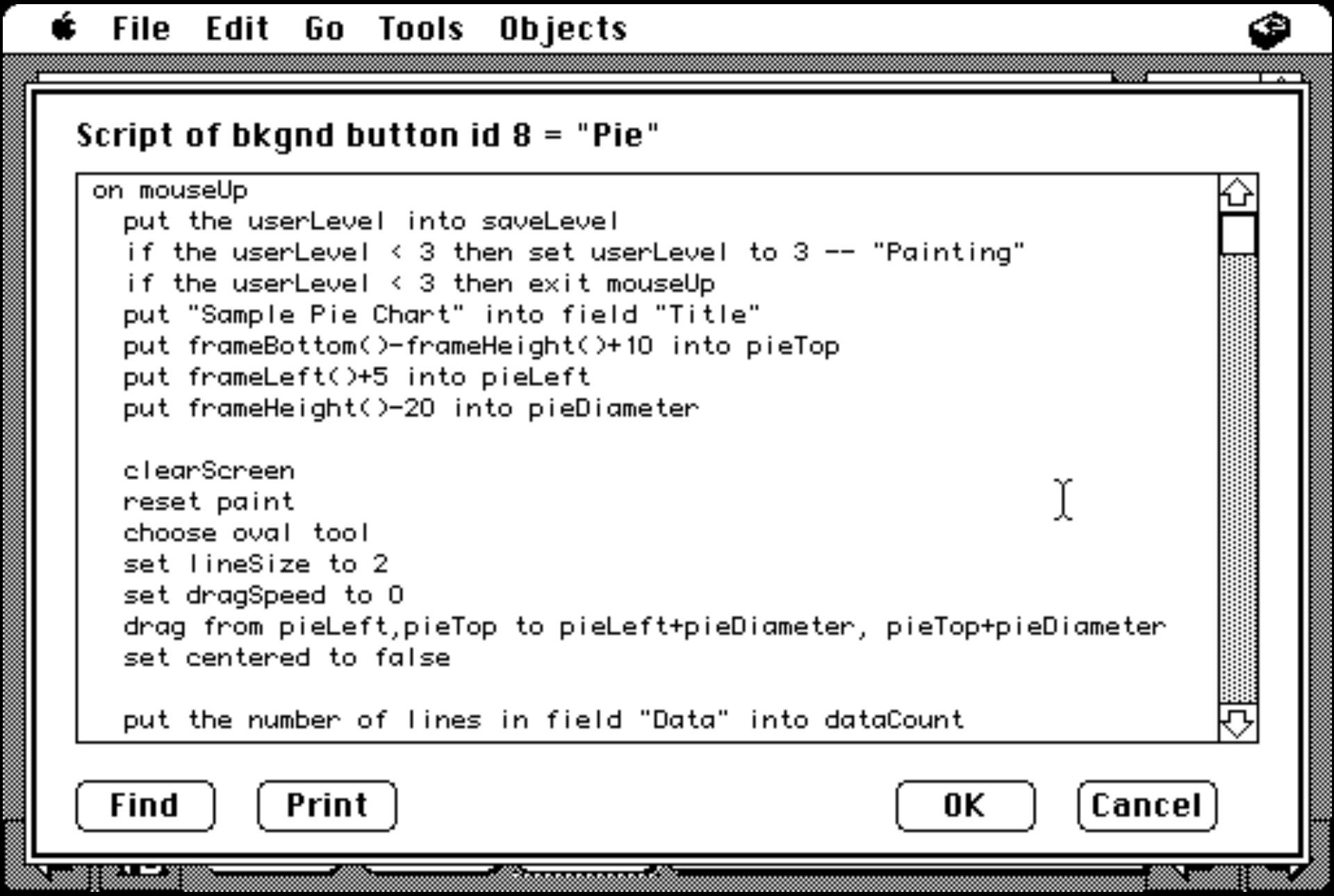

Quotations


Plots


1.2.5 Release Notes



8:06 AM


File Edit Go Tools Objects
Script of bkgnd button id 8 = "Pie"
on mouseUp
put the userLevel into saveLevel
if the userLevel < 3 then set userLevel to 3 -- "Painting"
if the userLevel < 3 then exit mouseUp
put "Sample Pie Chart" into field "Title"
put frameBottom()-frameHeight()+10 into pieTop
put frameLeft()+5 into pieLeft
put frameHeight()-20 into pieDiameter
clearScreen
reset paint
choose oval tool
set lineSize to 2
set dragSpeed to 0
drag from pieLeft,pieTop to pieLeft+pieDiameter, pieTop+pieDiameter
set centered to false
put the number of lines in field "Data" into dataCount
Find
Print
OK
Cancel

JavaScript is moving
to the server.

Servers

- Message driven, message queue
- Actor-like
- Simple events don’t fit:
  - Sequential
    - A sequence of requests, each dependent on the result of the previous.
    - Naïve approach: deeply nested handlers
  - Parallel
    - Do a bunch of independent things
    - Naïve approach: do it sequentially
  - Limited time, cancellation

Functional Programming to the Rescue

- Futures
  - Dataflow
- Promise
- Monads
- Arrows
- RX
- FRP: Flapjax, bacon.js, elm.

RQ

A JavaScript library for managing asynchronicity in server applications.

Four of five methods

`RQ.sequence(requestors)`

`RQ.parallel(requestors)`

`RQ.parallel(requestors, optionals)`

`RQ.race(requestors)`

`RQ.fallback(requestors)`

RQ.sequence

- Takes an array of requestor functions, calls them one at a time, passing the result of the previous requestor to the next requestor.

```
RQ.sequence([
    getUserData,
    getPreference(),
    getCustomNav()
])
```

RQ.parallel

- Takes an array of requestor functions, calls them all at once, and gives an array of results.

```
RQ.parallel([
    getNav(),
    getAds(),
    getMessageOfTheDay()
])
```

RQ.parallel

- Also takes an optional array of optional requestors. Their results will be included if they can be obtained before the required requestors finish.

```
RQ.parallel([
    getNav,
    getAds,
    getMessageOfTheDay
], [
    getHoroscope(),
    getGossip()
])
```

RQ.race

- Takes an array of requestors, calls them all at once, and gives the result of the first success.

```
getAds = RQ.race([
    getAd(adnet.KlikHaus),
    getAd(adnet.InUFace),
    getAd(adnet.TrackPipe)
]);
```

RQ.fallback

- Takes an array of requestors, and gives the result of the first success.

```
getWeather = RQ.fallback([
    fetch("weather", localCache),
    fetch("weather", localDB),
    fetch("weather", remoteDB)
]);
```

RQ	All at once	One at a time
All	`parallel`	`sequence`
One	`race`	`fallback`

RQ requestories with timeouts

`RQ.sequence(requestors, milliseconds)`

`RQ.parallel(requestors, milliseconds)`

`RQ.parallel(requestors, optionals, milliseconds, tilliseconds)`

`RQ.race(requestors, milliseconds)`

`RQ.fallback(requestors, milliseconds)`

Cancellation

- Any requestor can optionally return a quash function.
- A quash function, when called, will attempt to cancel a request.
- There is no guarantee that the cancellation will happen before the request completes.
- Cancellation is intended to stop unnecessary work. It does not undo.

RQ

- requestor

  A function that can execute a request
- requestion (requestor continuation)

  A continuation function that will be passed to a requestor
- requestory (requestor factory)

  A function that takes arguments and returns a requestor function.
- quash

  A function returned by a requestor that may be used to cancel a request.

requestor continuation

```
function requestion(success, failure)
```

quash

```
function quash(reason)
```

requestor

```
function requestor (
   function requestion (success, failure) ,
   value
) → function quash (reason)
```

requestor factory

```
function requestory(arguments…) →
    function requestor(
        function requestion(success, failure),
        value
    ) → function quash(reason)
```

Identity Requestor

```
function identity_requestor(
    requestion,
    value
) {
    requestion(value);
}
```

Fullname Requestor

```
function fullname_requestor(
    requestion,
    value
) {
    requestion(value.firstname
        + ' '
        + value.lastname);
}
```

Delay Requestor

```
function delay_requestor(
    requestion,
    value
) {
    var timeout_id = setTimeout(function () {
        requestion(value);
    }, 1000);
    return function quash(reason) {
        clearTimeout(timeout_id);
    };
}
```

Delay Requestory

```
function delay(milliseconds) {
    return function delay_requestor(
        requestion,
        value
    ) {
        var timeout_id = setTimeout(function () {
            requestion(value);
        }, milliseconds);
        return function quash(reason) {
            clearTimeout(timeout_id);
        };
    };
}
```

Simple Wrap Requestory

```
function wrap(func) {
    return function requestor(requestion, value) {
        requestion(func(value));
    };
}
```

```
function widget(name) {
    return function requestor(requestion, value) {
        var result = value ? value + '>' + name : name,
            fieldset = demo.tag('fieldset'),
            legend = demo.tag('legend')
                .value(name),
            success = demo.tag('input', 'button', 'success')
                .value('success')
                .on('click', function (e) {
                    fieldset.style('backgroundColor', 'lightgreen');
                    return requestion(result);
                }),
            failure = demo.tag('input', 'button', 'failure')
                .value('failure')
                .on('click', function (e) {
                    fieldset.style('backgroundColor', 'pink');
                    return requestion(undefined, result);
                });
        fieldset.append(legend);
        fieldset.append(success);
        fieldset.append(failure);
        demo.append(fieldset);
        return function quash() {
            fieldset.style('backgroundColor', 'silver');
        };
    };
}
```

```
function widget(name) {
    return function requestor(requestion, value) {
        var result = value ? value + '>' + name : name,
            fieldset = demo.tag('fieldset'),
            legend = demo.tag('legend')
                .value(name),
            success = demo.tag('input', 'button', 'success')
                .value('success')
                .on('click', function () {
                    fieldset.style('backgroundColor', 'lightgreen');
                    return requestion(result);
                }),
            failure = demo.tag('input', 'button', 'failure')
                .value('failure')
                .on('click', function () {
                    fieldset.style('backgroundColor', 'pink');
                    return requestion(undefined, result);
                });
        fieldset.append(legend);
        fieldset.append(success);
        fieldset.append(failure);
        demo.append(fieldset);
        return function quash() {
            fieldset.style('backgroundColor', 'silver');
        };
    };
}
```

```
function widget(name) {
    return function requestor(requestion, value) {
        var result = value ? value + '>' + name : name,
            fieldset = demo.tag('fieldset'),
            legend = demo.tag('legend')
                .value(name),
            success = demo.tag('input', 'button', 'success')
                .value('success')
                .on('click', function () {
                    fieldset.style('backgroundColor', 'lightgreen');
                    return requestion(result);
                }),
            failure = demo.tag('input', 'button', 'failure')
                .value('failure')
                .on('click', function () {
                    fieldset.style('backgroundColor', 'pink');
                    return requestion(undefined, result);
                });
        fieldset.append(legend);
        fieldset.append(success);
        return function quash() {
            fieldset.style('backgroundColor', 'silver');
        };
    };
}
```

```
RQ.parallel([
    RQ.sequence([
        widget('Seq A1'),
        widget('Seq A2'),
        widget('Seq A3')
    ]),
    RQ.sequence([
        widget('Seq B1'),
        widget('Seq B2'),
        widget('Seq B3')
    ]),
    widget('C'),
    RQ.race([
        widget('Race D1'),
        widget('Race D2'),
        widget('Race D3'),
    ]),
    RQ.fallback([
        widget('Fall E1'),
        widget('Fall E2'),
        widget('Fall E3')
    ])
], [
    RQ.sequence([
        widget('Opt Seq O1'),
        widget('Opt Seq O2'),
        widget('Opt Seq O3')
    ]),
    RQ.sequence([
        widget('Opt Seq P1'),
        widget('Opt Seq P2'),
        widget('Opt Seq P3')
    ]),
    widget('Opt Q'),
    RQ.race([
        widget('Opt Race R1'),
        widget('Opt Race R2'),
        widget('Opt Race R3'),
    ]),
    RQ.fallback([
        widget('Opt Fall S1'),
        widget('Opt Fall S2'),
        widget('Opt Fall S3')
    ])
])(show);
```

Testing

`assertEquals(message, expected, actual)`

does not work

QuickCheck

Koen Claessen
John Hughes
Chalmers University

JSCheck

Case generation

Testing over turns

- JSC.claim(name, predicate, signature)
- JSC.check(milliseconds)
- JSC.on_report(callback)
- JSC.on_error(callback)

`JSC.claim(`name, predicate, signature`)`

- name is a string
- **`function`** predicate`(`verdict`,` et al…`)`
- signature is an array of specifications, one per et al…

```
JSC.claim(
    "Compare the old code with the new code",
    function predicate(verdict, a) {
        verdict(oldCode(a) === newCode(a));
    },
    [JSC.integer()]
);
```

Specifiers

```
JSC.any()
JSC.array()
JSC.boolean()
JSC.character()
JSC.falsy()
JSC.integer()
JSC.literal()
JSC.number()
JSC.object()
JSC.one_of()
JSC.sequence()
JSC.string()
```

```
JSC.string(
    3, JSC.character('0', '9'),
    1, '-',
    2, JSC.character('0', '9'),
    1, '-',
    4, JSC.character('0', '9')
)

"094-55-0695"
"571-63-9387"
"130-08-5751"
"296-55-3384"
"976-55-3359"
```

```
JSC.array([
    JSC.integer(),
    JSC.number(100),
    JSC.string(8, JSC.character('A', 'Z'))
])

[3,  21.228644298389554, "TJFJPLQA"]
[5,  57.05485427752137,  "CWQDVXWY"]
[7,  91.98980208020657,  "QVMGNVXK"]
[11, 87.07735128700733,  "GXBSVLKJ"]
```

```
JSC.object({
    left: JSC.integer(640),
    top: JSC.integer(480),
    color: JSC.one_of([
        'black', 'white', 'red',
        'blue', 'green', 'gray'
    ])
})
```

```
{"left":104, "top":139, "color":"gray"}
{"left":62,  "top":96,  "color":"white"}
{"left":501, "top":164, "color":"red"}
{"left":584, "top":85,  "color":"white"}
```

```
JSC.object(
    JSC.array(
        JSC.integer(3, 8),
        JSC.string(4, JSC.character('a', 'z'))
    ),
    JSC.boolean()
)
```

```
{"jodo":true, "zhzm":false, "rcqz":true}
{"odcr":true, "azax":true,  "bnfx":true,
    "hmmc":false}
{"wjew":true, "kgqj":true,  "abid":true,
    "cjva":false, "qsgj":true, "wtsu":true}
{"qtbo":false, "vqzc":false, "zpij":true,
    "ogss":false, "lxnp":false, "psso":true,
    "irha":true,  "ghnj":true}
```

verdict

- When check calls a predicate, it passes in a verdict function.
- Predicates deliver the result of each trial by calling the verdict function.
- verdict is a continuation, allowing trials to extend over many turns.
- Three outcomes:

pass fail lost

Closure.
Continuation.

https://github.com/douglascrockford/RQ

https://github.com/douglascrockford/JSCheck