
Google™
10

Beyond JavaScript:
Programming the Web With Native Code

David Springer
Ian Lewis
May 19,2010

View live notes and ask questions about this session on Google Wave:

http://bit.ly/cgOGiI

Agenda

- Overview
- The Native Client SDK
- Tutorial
- Coming attractions

http://bit.ly/cgOGil

Non-agenda

- Internals and technical details
- Security model

First Things First…

Native Client

NaCl

Overview: What Is NaCl?

(And Why Should I Care?)

Overview: What Is NaCl?

- A portable system for **verifying** and executing **untrusted native** code in the browser
  - High performance like C++
  - Safe and portable like JavaScript
- A runtime sandbox
  - "Virtual Mini-POSIX"
  - Standard operating system services
    - Kernel
    - Window manager
    - Media layer
  - Identical on every supported platform

NaCl: What Is It Good For?

- Port desktop applications to the web
  - Zero install
  - Native performance
- Enhance web apps with...
  - Existing C/C++ libraries (libcrypt, CGAL, etc)
  - New high-performance compiled code
- Sandbox existing plugin code
  - Stop asking users to trust your code

Lunch Ain't Free

- Must compile **verifiable** code
  - Minor performance penalty
  - Generally larger executables
- Some *nix syscalls are unavailable
  - Process creation
  - Direct network/file access
- Still some rough edges

How Does This Make My Life Better?

How Does This Make My Life Better?

How Does This Make My Life Better?

- Native performance

How Does This Make My Life Better?

- Native performance
- Platform-independent multimedia

How Does This Make My Life Better?

- Native performance
- Platform-independent multimedia
- Your choice of language

How Does This Make My Life Better?

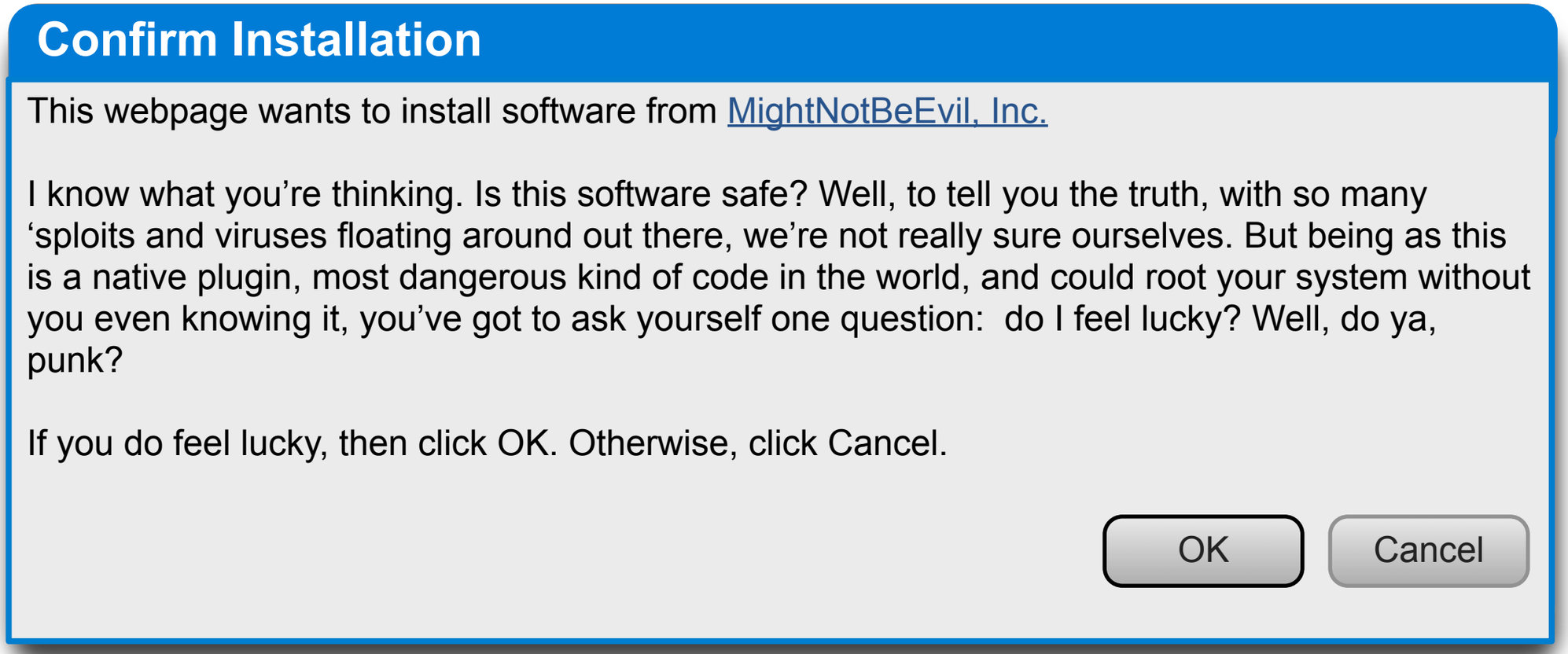
- Native performance
- Platform-independent multimedia
- Your choice of language
- Low-level system services

How Does This Make My Life Better?

- Native performance
- Platform-independent multimedia
- Your choice of language
- Low-level system services
- The end of:

How Does This Make My Life Better?

- Native performance
- Platform-independent multimedia
- Your choice of language
- Low-level system services
- The end of:



Demo: Unity

The NaCl SDK

SDK Goodies

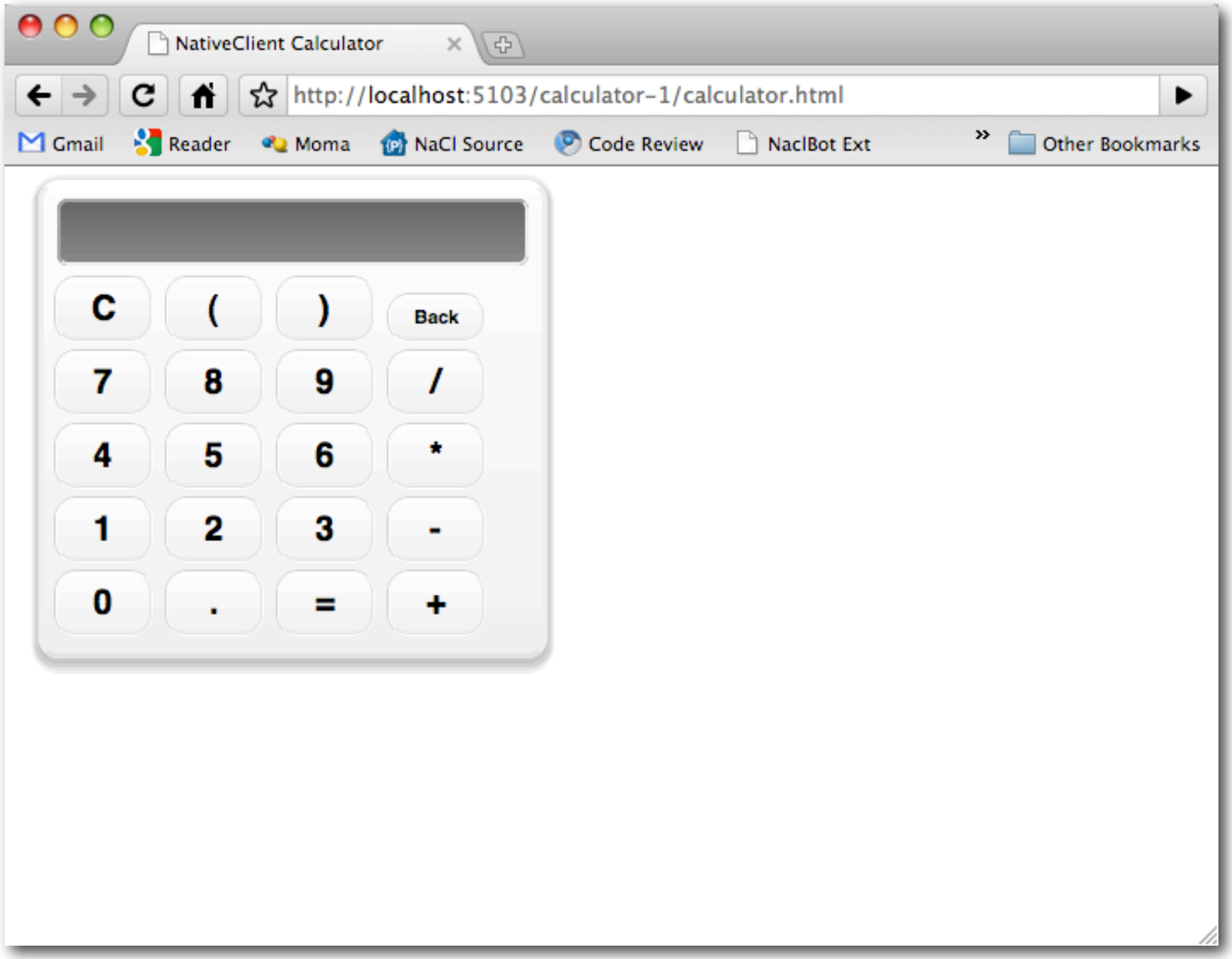
- C/C++ toolchain
- Standard GNU libraries
- “Pepper”
  - NPAPI-like interface
  - Audio / video
  - OpenGL ES 2.0
  - Javascript interop
- Sample code + build scripts

Using the SDK

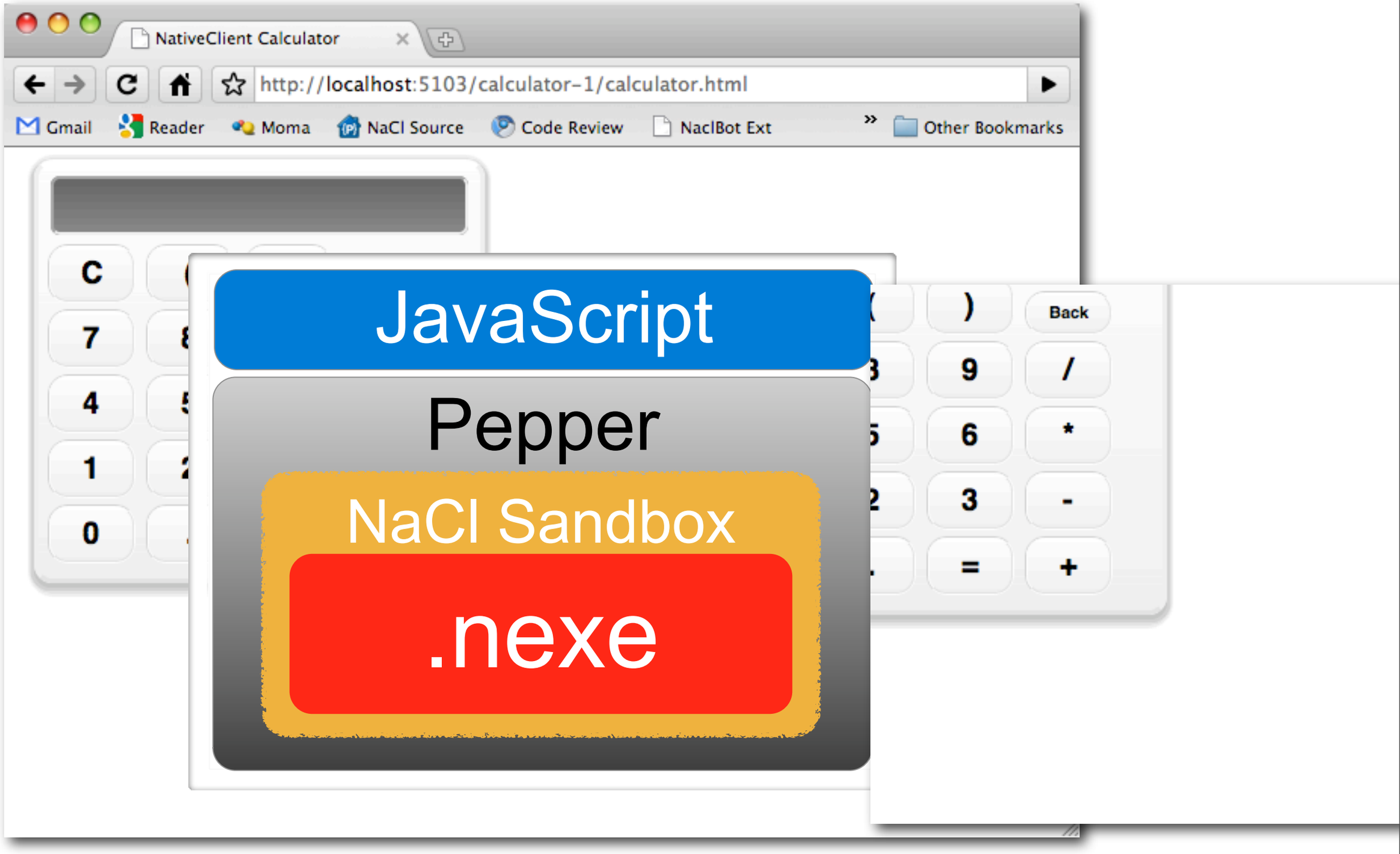
- Build with standard GNU toolchain
  - Tools run on Mac, Linux, Windows
  - Output is OS-independent
- Run in web browser
  - Server required
- Debug with GDB-compatible debugger
  - Via GDB stub

Calculator Tutorial

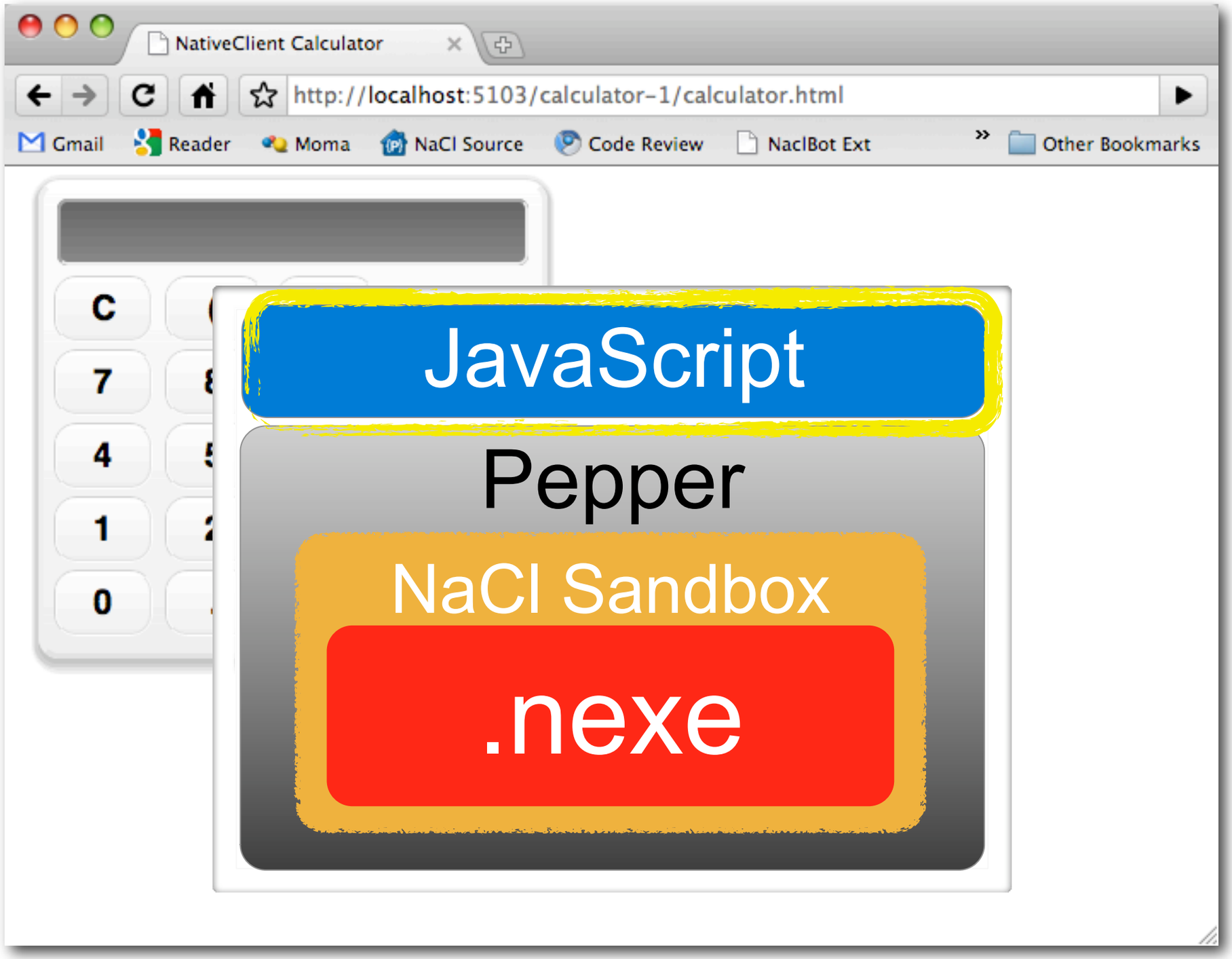
Anatomy of a NaCl Application



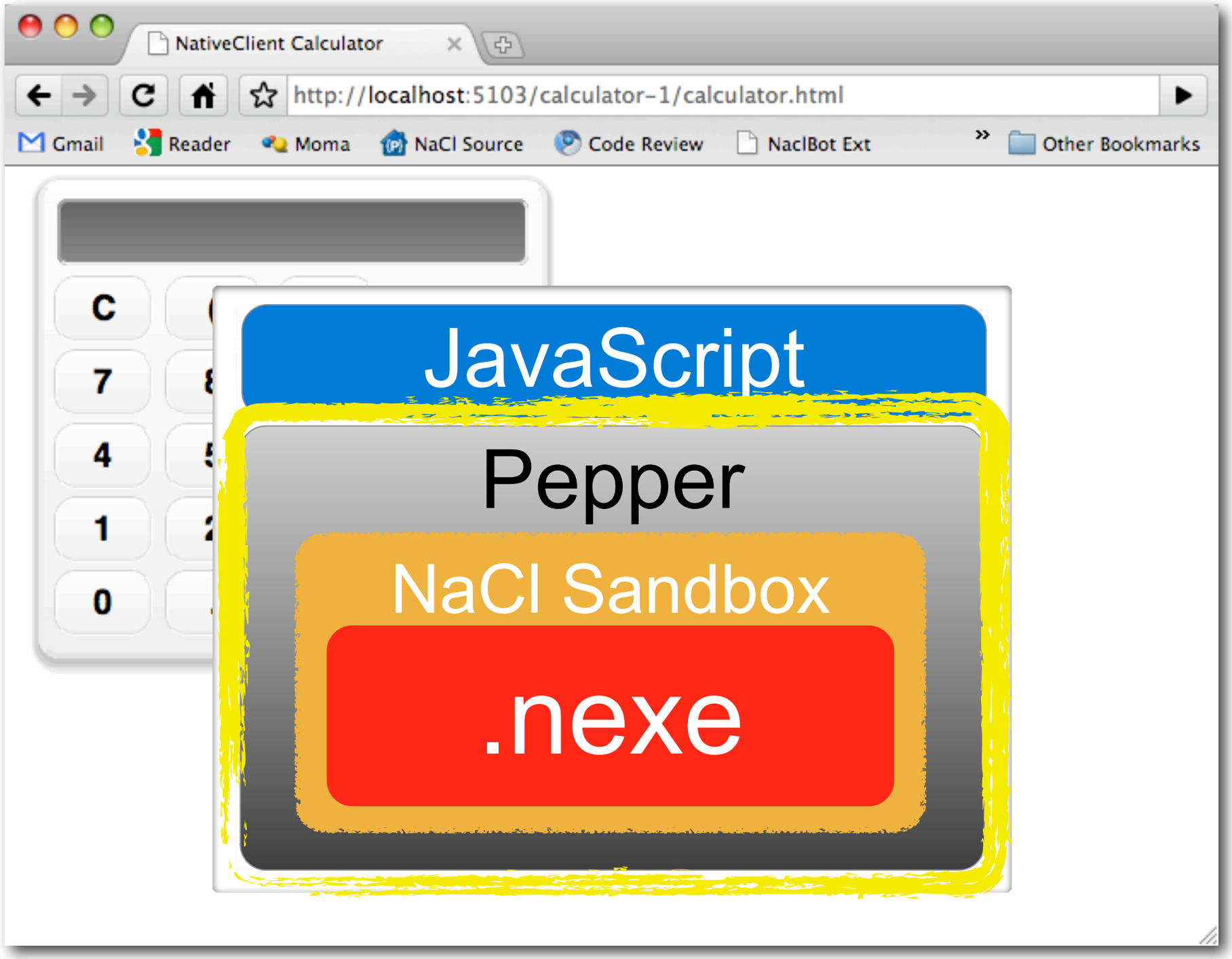
Anatomy of a NaCl Application



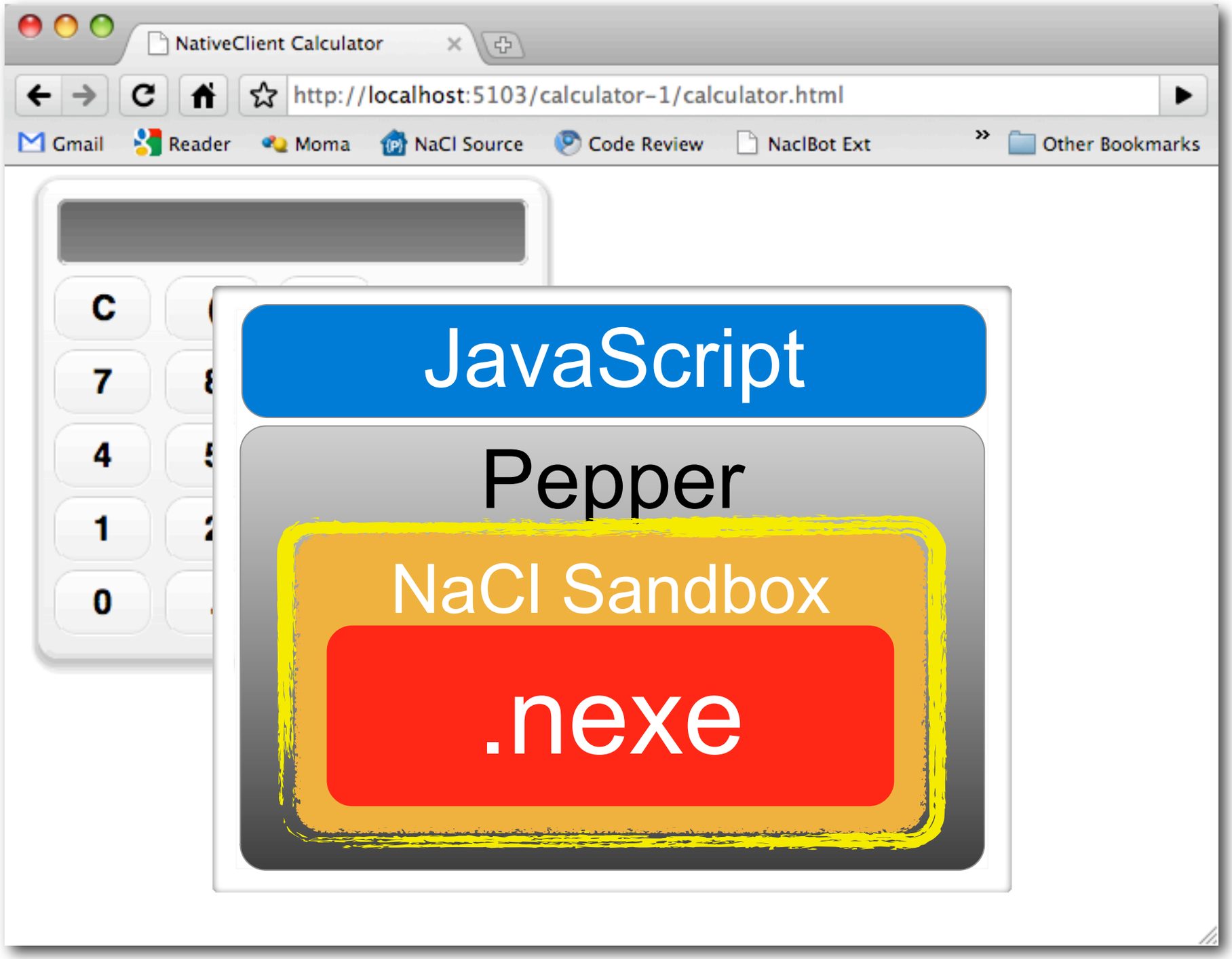
Anatomy of a NaCl Application



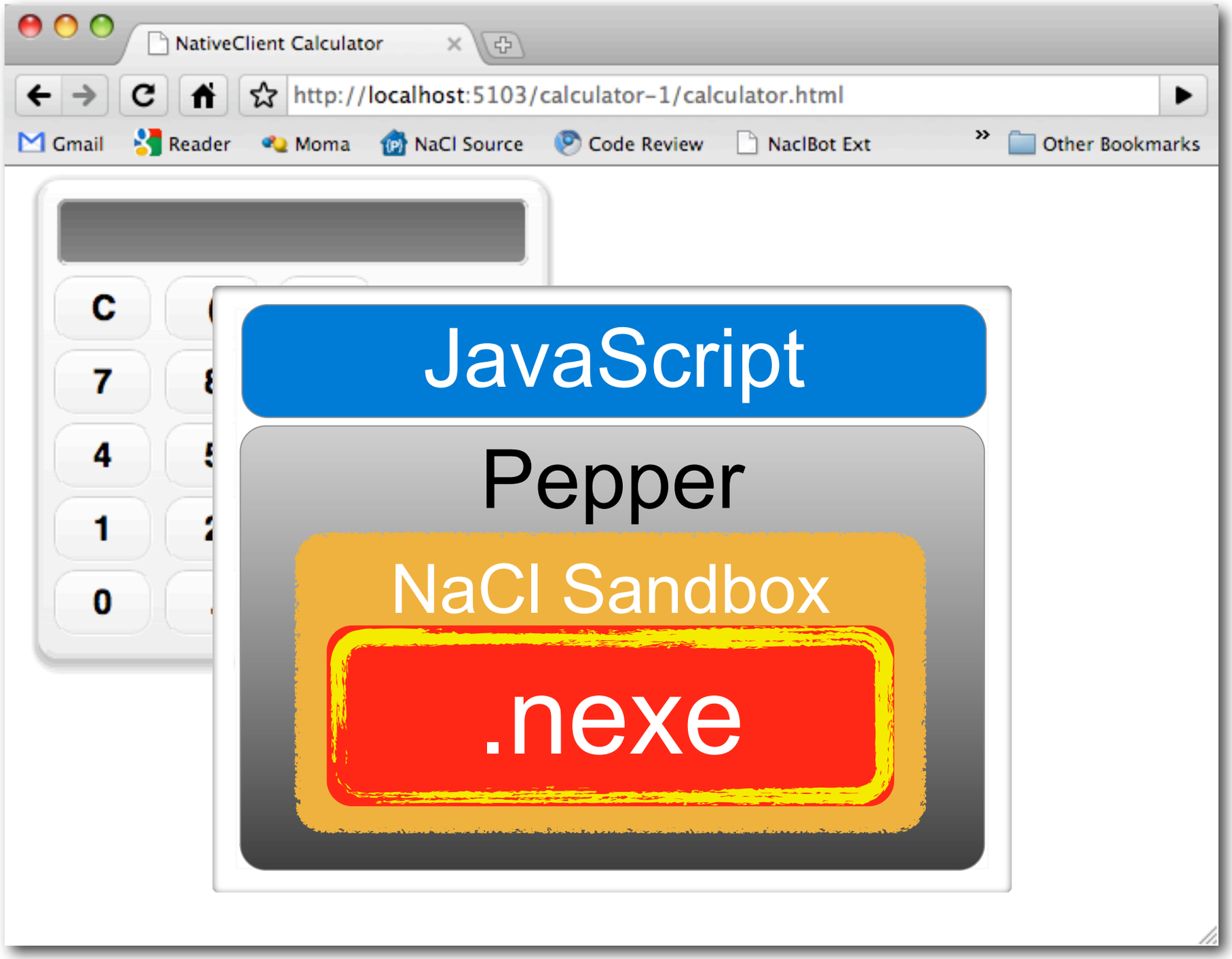
Anatomy of a NaCl Application



Anatomy of a NaCl Application



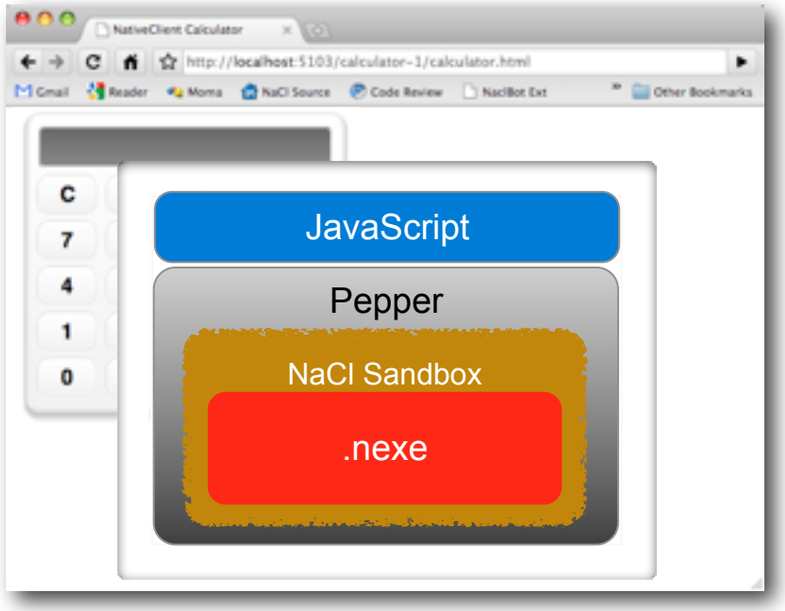
Anatomy of a NaCl Application



Tutorial: Getting Started

- Install the SDK
  - `/usr/local/nacl-sdk`
- Organize project directories and files:
  - Tutorial layout:

```
application/
   Browser code: *.html, *.js

nacl/
   Module-specific code: *.cc

c_salt/
   Bridging: Interop between *.cc and *.js
```



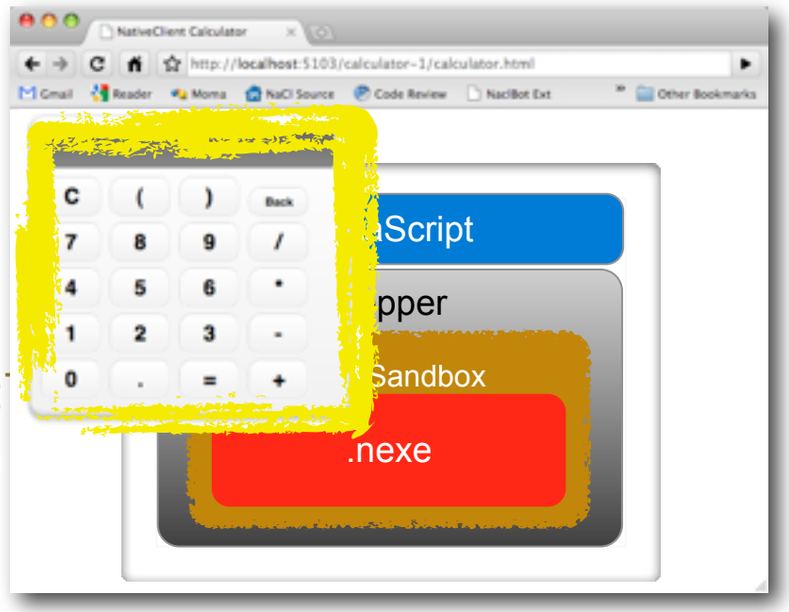
Tutorial: Adding UI

HTML + CSS

```
<body onload="init()">

  <div class="shadow-out"><div class="shadow-mid"><div class="shadow-in">
    <!-- For development, use a #develop location, which loads the develop
    version of the module.
    -->
    <div id="calculator_content"></div>
    <script type="text/javascript">
    </script>
    <div id="calculator">
      <input type="text" id="formula" />

      <div>
        <div class="button"><span onclick="clean()">C</span></div>
        <div class="button"><span onclick="pressOperator(this)">(</span></div>
        <div class="button"><span onclick="pressOperator(this)">)</span></div>
        <div class="button"><span onclick="back()"><b style="font-size: 11px">Back</b>
            </span></div>
      </div>

      <div>
        <div class="button"><span onclick="pressNumber(this)">7</span></div>
        <div class="button"><span onclick="pressNumber(this)">8</span></div>
        <div class="button"><span onclick="pressNumber(this)">9</span></div>
        <div class="button"><span onclick="pressOperator(this)">/</span></div>
```

Tutorial: Adding UI

HTML + CSS




```
<body onload="init()">

  <div class="shadow-out"><div class="shadow-mid"><div c
    <!-- For development, use a #develop location, which
    version of the module.
    -->
    <div id="calculator_content"></div>
    <script type="text/javascript">
    </script>
    <div id="calculator">
      <input type="text" id="formula" />

      <div>
        <div class="button"><span onclick="clean()">C</span></div>
        <div class="button"><span onclick="pressOperator(this)">(</span></div>
        <div class="button"><span onclick="pressOperator(this)">)</span></div>
        <div class="button"><span onclick="back()"><b style="font-size: 11px">Back</b>
            </span></div>
      </div>

      <div>
        <div class="button"><span onclick="pressNumber(this)">7</span></div>
        <div class="button"><span onclick="pressNumber(this)">8</span></div>
        <div class="button"><span onclick="pressNumber(this)">9</span></div>
        <div class="button"><span onclick="pressOperator(this)">/</span></div>
```

Tutorial: Adding UI

HTML + CSS

```
<body onload="init()">

  <div class="shadow-out"><div class="shadow-mid"><div class="shadow-in">
    <!-- For development, use a #develop location, which loads the develop
    version of the module.
    -->
    <div id="calculator_content"></div>
    <script type="text/javascript">
    </script>
    <div id="calculator">
      <input type="text" id="formula" />

      <div>
        <div class="button"><span onclick="clean()">C</span></div>
        <div class="button"><span onclick="pressOperator(this)">(</span></div>
        <div class="button"><span onclick="pressOperator(this)">)</span></div>
        <div class="button"><span onclick="back()"><b style="font-size: 11px">Back</b>
            </span></div>
      </div>

      <div>
        <div class="button"><span onclick="pressNumber(this)">7</span></div>
        <div class="button"><span onclick="pressNumber(this)">8</span></div>
        <div class="button"><span onclick="pressNumber(this)">9</span></div>
        <div class="button"><span onclick="pressOperator(this)">/</span></div>
```

Tutorial: Adding UI

HTML + CSS

```
.button span {
  text-align: center;
  color: black;
  font-family: sans-serif;
  font-weight: bold;
  display: block;
  border-right: 1px solid #ddd;
  border-left: 1px solid #fff;
  border-top: 1px solid #fff;
  border-bottom: 1px solid #ddd;
  width: 30px;
  border-radius: 12px;
  -moz-border-radius: 12px;
  -webkit-border-radius: 12px;
  background: -webkit-gradient(linear, left top, left bottom, from(#fff), to(#eee));
  background: -moz-linear-gradient(top, #fff, #eee);
  padding: 6px 12px 6px 12px;
  font-size: 20px;
  text-decoration: none;
  float: left;
}

.zbutton span:hover {
  border: 3px solid #f00;
```

Tutorial: Adding UI

HTML + CSS

```
      <div class="button"><span onclick="pressNumber(this)">1</span></div>
      <div class="button"><span onclick="pressNumber(this)">2</span></div>
      <div class="button"><span onclick="pressNumber(this)">3</span></div>
      <div class="button"><span onclick="pressOperator(this)">-</span></div>
    </div>

    <div>
      <div class="button"><span onclick="pressNumber(this)">0</span></div>
      <div class="button"><span onclick="pressNumber(this)">.</span></div>
      <div class="button"><span onclick="startCalculate()">=</span></div>
      <div class="button"><span onclick="pressOperator(this)">+</span></div>
    </div>
  </div>
</div></div></div>

<style>

body {
  text-align: center;
  margin-top: 20%;
}

.shadow-in {
  display: inline-block;
  border-radius: 13px;
```

Tutorial: Adding UI

HTML + CSS

```
      <div class="button"><span onclick="pressNumber(this)">1</span></div>
      <div class="button"><span onclick="pressNumber(this)">2</span></div>
      <div class="button"><span onclick="pressNumber(this)">3</span></div>
      <div class="button"><span onclick="pressOperator(this)">-</span></div>
    </div>

    <div>
      <div class="button"><span onclick="pressNumber(this)">0</span></div>
      <div class="button"><span onclick="pressNumber(this)">.</span></div>
      <div class="button"><span onclick="startCalculate()">=</span></div>
      <div class="button"><span onclick="pressOperator(this)">+</span></div>
    </div>
  </div>
</div></div></div>

<style>

body {
  text-align: center;
  margin-top: 20%;
}

.shadow-in {
  display: inline-block;
  border-radius: 13px;
```

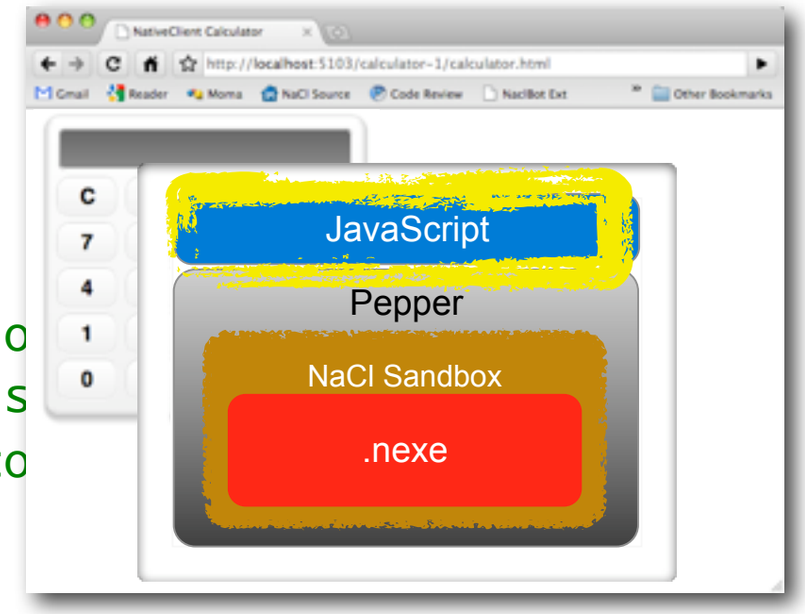
Tutorial: Adding UI

HTML + CSS + JavaScript

```
/**
 * Here we start a calculation, calling a custom method on the nexe that
 * has an asynchronous callback. (The API also supports synchronous method
 * calls for things that are really fast. This example could, of course,
 * be handled synchronously.)
 */
function startCalculate() {
  // Convert the formula to postfix notation, and put the postfix expression
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
```

Tutorial: Adding UI

HTML + CSS + JavaScript




```
/**
 * Here we start a calculation, calling a custom method o
 * has an asynchronous callback. (The API also supports s
 * calls for things that are really fast. This example co
 * be handled synchronously.)
 */
function startCalculate() {
  // Convert the formula to postfix notation, and put the postfix expression
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
```

Google I/O 10

Tutorial: Adding UI

HTML + CSS + JavaScript

```
/**
 * Here we start a calculation, calling a custom method on the nexe that
 * has an asynchronous callback. (The API also supports synchronous method
 * calls for things that are really fast. This example could, of course,
 * be handled synchronously.)
 */
function startCalculate() {
  // Convert the formula to postfix notation, and put the postfix expression
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
```

Tutorial: Adding UI

HTML + CSS + JavaScript

```
var operator_stack = [];
while (infix.offset < infix.string.length) {
  // Consume tokens until the end of the input string is reached.
  var token = google.expr.nextToken_(infix);
  if (!token || token.type == google.expr.TokenType.NOTYPE) {
    throw new SyntaxError("Unrecognized token.");
  }
  switch (token.type) {
  case google.expr.TokenType.NUMBER:
    rpn_out.push(token.token);
    break;
  case google.expr.TokenType.OPERATOR:
    while (operator_stack.length > 0) {
      // Only handle left-associative operators.  Pop off all the operators
      // that have less or equal precedence to |token|.
      var operator = operator_stack.pop();
      if (operator.type == google.expr.TokenType.OPERATOR &&
          operator.precedence <= token.precedence) {
        rpn_out.push(operator.token);
      } else {
        // The top-of-stack operator has a higher precedence than the current
        // token, or it's a parenthesis.  Push it back on the stack and stop.
        operator_stack.push(operator);
        break;
      }
```

Tutorial: Adding UI

HTML + CSS + JavaScript

```
/**
 * Here we start a calculation, calling a custom method on the nexe that
 * has an asynchronous callback. (The API also supports synchronous method
 * calls for things that are really fast. This example could, of course,
 * be handled synchronously.)
 */
function startCalculate() {
  // Convert the formula to postfix notation, and put the postfix expression
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
```

Tutorial: Adding UI

HTML + CSS + JavaScript

```
/**
 * Here we start a calculation, calling a custom method on the nexe that
 * has an asynchronous callback. (The API also supports synchronous method
 * calls for things that are really fast. This example could, of course,
 * be handled synchronously.)
 */
function startCalculate() {
  // Convert the formula to postfix notation, and put the postfix expression
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
```

Tutorial: Adding UI

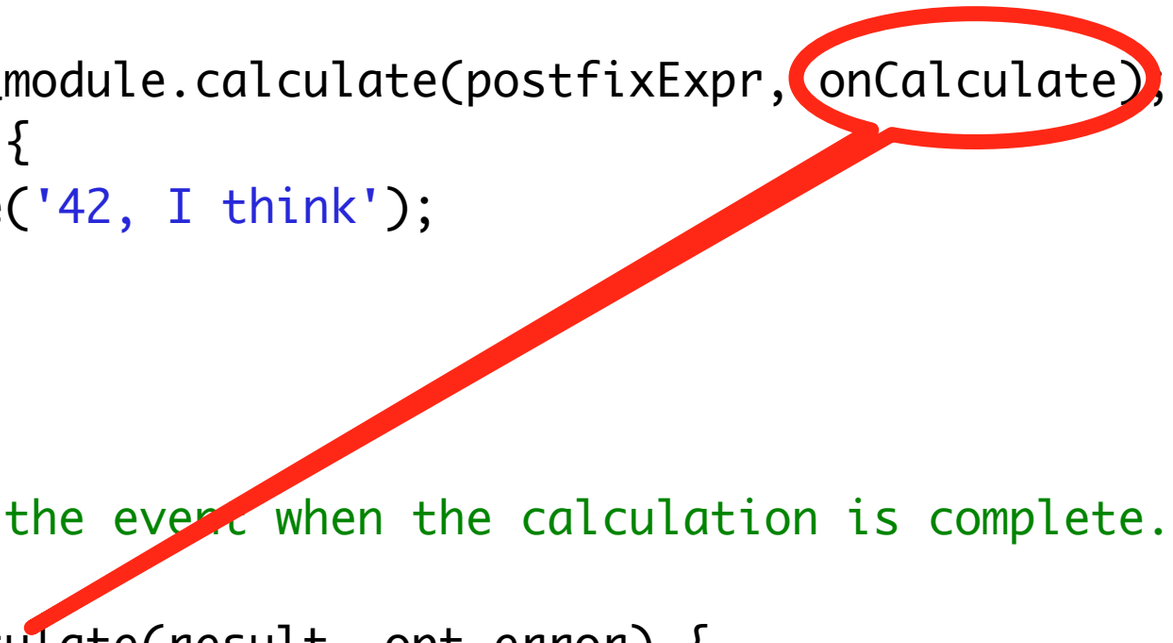
HTML + CSS + JavaScript

```
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
  } else {
    document.getElementById('formula').value = result;
  }
}
```

Tutorial: Adding UI

HTML + CSS + JavaScript

```
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
  } else {
    document.getElementById('formula').value = result;
  }
}
```



Calculator Tutorial Step 1

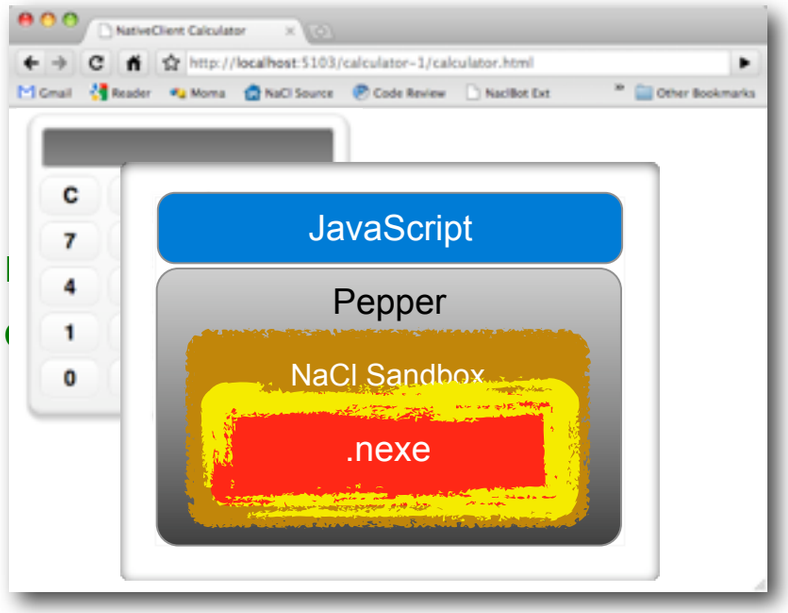
Tutorial: Adding Methods

C++

```
// Copyright 2010 The Native Client Authors. All rights reserved.
// Use of this source code is governed by a BSD-style license that can
// be found in the LICENSE file.

#include "nacl/calculator.h"

#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#if defined(__native_client__)
#include <nacl/nacl_imc.h>
#include <nacl/nacl_npapi.h>
#include <nacl/npapi_extensions.h>
#include <nacl/npruntime.h>
#else
// Building the develop version.
#include "third_party/npapi/bindings/npapi.h"
#include "third_party/npapi/bindings/npapi_extensions.h"
#include "third_party/npapi/bindings/nphostapi.h"
#endif

#include "c_sqlt/double_type.h"
```

Tutorial: Adding Methods

C++




```
// Copyright 2010 The Native Client Authors. All rights r
// Use of this source code is governed by a BSD-style lic
// be found in the LICENSE file.

#include "nacl/calculator.h"

#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#if defined(__native_client__)
#include <nacl/nacl_imc.h>
#include <nacl/nacl_npapi.h>
#include <nacl/npapi_extensions.h>
#include <nacl/npruntime.h>
#else
// Building the develop version.
#include "third_party/npapi/bindings/npapi.h"
#include "third_party/npapi/bindings/npapi_extensions.h"
#include "third_party/npapi/bindings/nphostapi.h"
#endif

#include "c_sqlt/double_type.h"
```

Google I/O 10

Tutorial: Adding Methods

C++

```
// Copyright 2010 The Native Client Authors. All rights reserved.
// Use of this source code is governed by a BSD-style license that can
// be found in the LICENSE file.

#include "nacl/calculator.h"

#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#if defined(__native_client__)
#include <nacl/nacl_imc.h>
#include <nacl/nacl_npapi.h>
#include <nacl/npapi_extensions.h>
#include <nacl/npruntime.h>
#else
// Building the develop version.
#include "third_party/npapi/bindings/npapi.h"
#include "third_party/npapi/bindings/npapi_extensions.h"
#include "third_party/npapi/bindings/nphostapi.h"
#endif

#include "c_sqlt/double_type.h"
```

Tutorial: Adding Methods

C++

```
#include "c_salt/double_type.h"
#include "c_salt/int32_type.h"
#include "c_salt/object_type.h"
#include "c_salt/scripting_bridge.h"
#include "c_salt/string_type.h"

namespace c_salt {

// The required factory method.
Module* Module::CreateModule() {
  return new calculator::Calculator();
}

}  // namespace c_salt

namespace calculator {

Calculator::Calculator()
    : calculate_callback_(NULL) {
}

Calculator::~Calculator() {
  delete calculate_callback_;
}
```

Tutorial: Adding Methods

C++

```
#include "c_salt/double_type.h"
#include "c_salt/int32_type.h"
#include "c_salt/object_type.h"
#include "c_salt/scripting_bridge.h"
#include "c_salt/string_type.h"

namespace c_salt {

// The required factory method.
Module* Module::CreateModule() {
  return new calculator::Calculator();
}

}  // namespace c_salt

namespace calculator {

Calculator::Calculator()
    : calculate_callback_(NULL) {
}

Calculator::~Calculator() {
  delete calculate_callback_;
}
```

Tutorial: Adding Methods

C++

```
#include "c_salt/double_type.h"
#include "c_salt/int32_type.h"
#include "c_salt/object_type.h"
#include "c_salt/scripting_bridge.h"
#include "c_salt/string_type.h"

namespace c_salt {

// The required factory method.
Module* Module::CreateModule() {
  return new calculator::Calculator();
}

}  // namespace c_salt

namespace calculator {

Calculator::Calculator()
    : calculate_callback_(NULL) {
}

Calculator::~Calculator() {
  delete calculate_callback_;
}
```

Tutorial: Adding Methods

C++

```
Calculator::~Calculator() {
  delete calculate_callback_;
}

void Calculator::InitializeMethods(c_salt::ScriptingBridge* bridge) {
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
}
```

```
bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
```

Tutorial: Adding Methods

C++

```
Calculator::~Calculator() {
  delete calculate_callback_;
}

void Calculator::InitializeMethods(c_salt::ScriptingBridge* bridge) {
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
}
```

```
bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
```

Tutorial: Adding Methods

C++

```
Calculator::~Calculator() {
  delete calculate_callback_;
}

void Calculator::InitializeMethods(c_salt::ScriptingBridge* bridge) {
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
}
```

```
bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
```

Tutorial: Adding Methods

C++

```
Calculator::~Calculator() {
  delete calculate_callback_;
}

void Calculator::InitializeMethods(c_salt::ScriptingBridge* bridge) {
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
```

Tutorial: Adding Methods

C++

```
Calculator::~Calculator() {
  delete calculate_callback_;
}

void Calculator::InitializeMethods(c_salt::ScriptingBridge* bridge) {
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
}
```

```
bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
```

Tutorial: Adding Methods

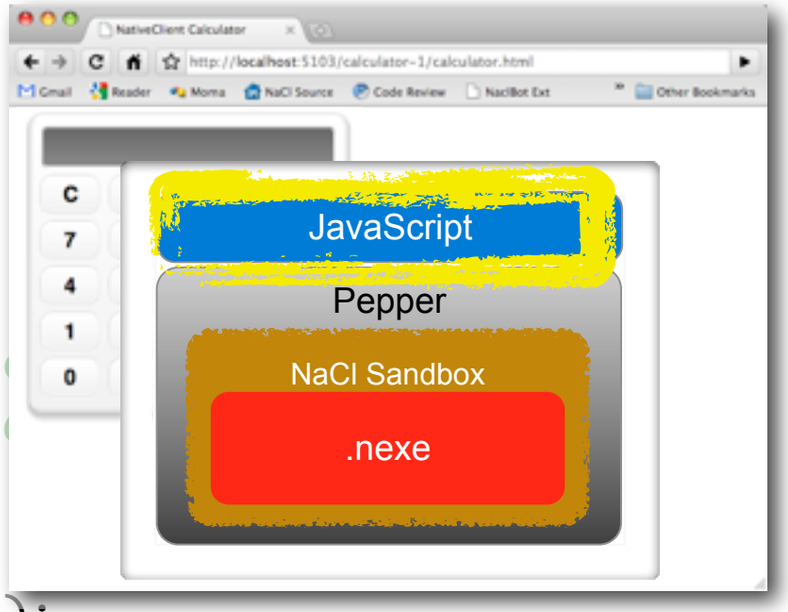
JavaScript

```
 */
function startCalculate() {
  // Convert the formula to postfix notation, and put the postfix expression
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
  } else {
    document.getElementById('formula').value = result;
  }
}
```

Tutorial: Adding Methods

JavaScript

```
 */
function startCalculate() {
  // Convert the formula to postfix notation, and put th
  // into an array that gets passed to the calculator mo
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
  } else {
    document.getElementById('formula').value = result;
  }
}
```



Tutorial: Adding Methods

JavaScript

```
 */
function startCalculate() {
  // Convert the formula to postfix notation, and put the postfix expression
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
  } else {
    document.getElementById('formula').value = result;
  }
}
```

Tutorial: Adding Methods

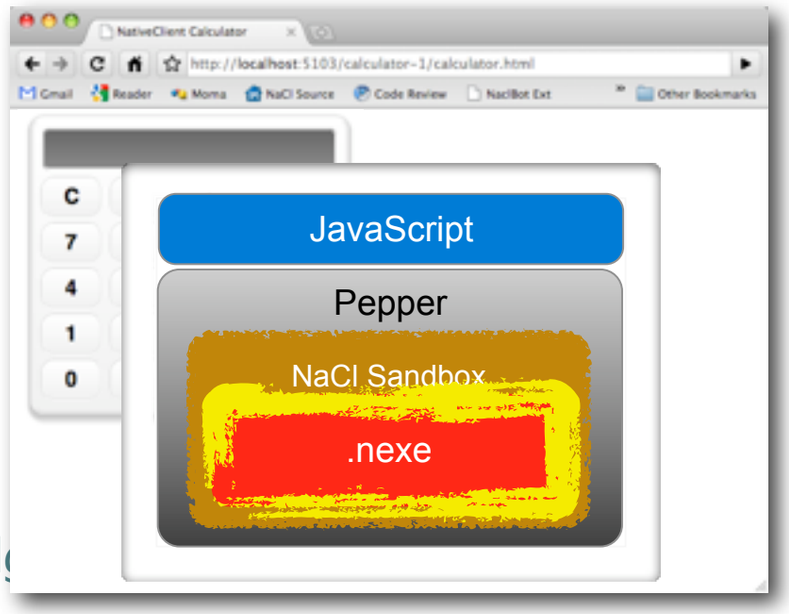
C++

```
Calculator::~Calculator() {
  delete calculate_callback_;
}

void Calculator::InitializeMethods(c_salt::ScriptingBridge* bridge) {
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
}
```

```
bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
```

Tutorial: Adding Methods

C++




```
Calculator::~Calculator() {
  delete calculate_callback_;
}

void Calculator::InitializeMethods(c_salt::ScriptingBrid
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
}
```

```
bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
```

Tutorial: Adding Methods

C++

```
Calculator::~Calculator() {
  delete calculate_callback_;
}

void Calculator::InitializeMethods(c_salt::ScriptingBridge* bridge) {
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
}
```

```
bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
```

Tutorial: Adding Methods

C++

```
                                NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
    c_salt::ObjectType function_obj(NPVARIANT_TO_OBJECT(args[1]));
    // Pack the value of the expression into the first arg of the callback
    // function, and invoke it.
    NPVariant argv;
    NPVariant result;
    DOUBLE_TO_NPVARIANT(expr_value, argv);
    NULL_TO_NPVARIANT(result);
    NPN_InvokeDefault(bridge->npp(),
                      function_obj.object_value(),
                      &argv,
                      1,
                      &result);
}
```

Tutorial: Adding Methods

C++

```
                              NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
    c_salt::ObjectType function_obj(NPVARIANT_TO_OBJECT(args[1]));
    // Pack the value of the expression into the first arg of the callback
    // function, and invoke it.
    NPVariant argv;
    NPVariant result;
    DOUBLE_TO_NPVARIANT(expr_value, argv);
    NULL_TO_NPVARIANT(result);
    NPN_InvokeDefault(bridge->npp(),
                      function_obj.object_value(),
                      &argv,
                      1,
                      &result);
  }
```

Tutorial: Adding Methods

C++

```
double Calculator::EvaluateExpression(
    const c_salt::Type::TypeArray* expression) {
  if (expression->size() == 0)
    return 0.0;
  // This is a pretty simple-minded expression evaulator.  It assumes that the
  // input vector represents a correct postfix expression and does basically
  // no error checking.  If the input vector is badly formed, then you will get
  // unpredicatble results.
  std::vector<double> expr_stack;
  std::vector<c_salt::Type*>::const_iterator it;
  for (it = expression->begin(); it != expression->end(); ++it) {
    if ((*it)->type_id() == c_salt::Type::kStringTypeId) {
      std::string* oper =
          (static_cast<c_salt::StringType*>(*it))->string_value();
      double term0, term1;
      switch (oper->at(0)) {
      case '+':
        assert(expr_stack.size() >= 2);
        term0 = expr_stack.back();
        expr_stack.pop_back();
        term1 = expr_stack.back();
        expr_stack.pop_back();
        expr_stack.push_back(term1 + term0);
        break;
```

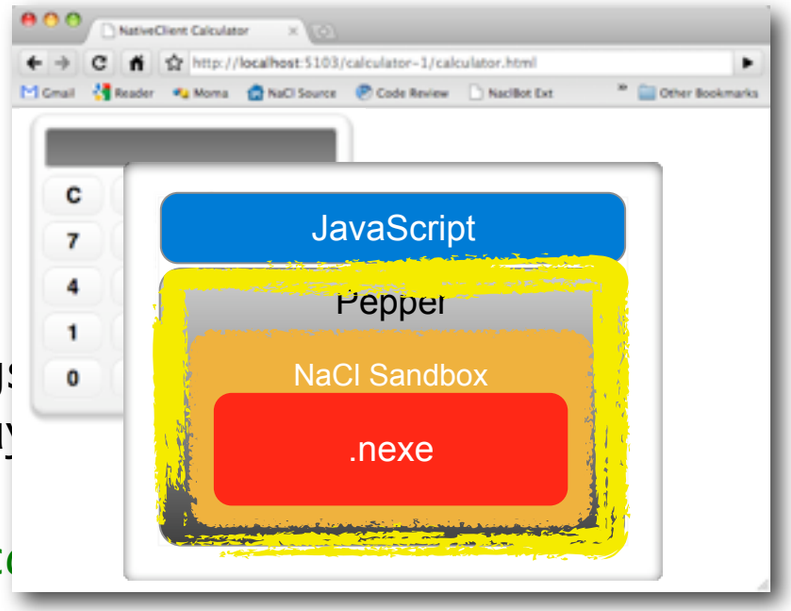
Tutorial: Adding Methods

C++

```
  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
    c_salt::ObjectType function_obj(NPVARIANT_TO_OBJECT(args[1]));
    // Pack the value of the expression into the first arg of the callback
    // function, and invoke it.
    NPVariant argv;
    NPVariant result;
    DOUBLE_TO_NPVARIANT(expr_value, argv);
    NULL_TO_NPVARIANT(result);
    NPN_InvokeDefault(bridge->npp(),
                      function_obj.object_value(),
                      &argv,
                      1,
                      &result);
  }

  return true;
}
```

Tutorial: Adding Methods

C++




```
c_salt::Type::TypeArray* expression_array =
    c_salt::Type::CreateArrayFromNPVariant(bridge, args
double expr_value = EvaluateExpression(expression_array

// If there was a second argument, assume it's the once
if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
  // The ObjectType ctor bumps the ref count of the callback function.
  c_salt::ObjectType function_obj(NPVARIANT_TO_OBJECT(args[1]));
  // Pack the value of the expression into the first arg of the callback
  // function, and invoke it.
  NPVariant argv;
  NPVariant result;
  DOUBLE_TO_NPVARIANT(expr_value, argv);
  NULL_TO_NPVARIANT(result);
  NPN_InvokeDefault(bridge->npp(),
                    function_obj.object_value(),
                    &argv,
                    1,
                    &result);
}

return true;
}
```

Tutorial: Adding Methods

C++

```
  c_salt::Type::TypeArray* expression_array =
      c_salt::Type::CreateArrayFromNPVariant(bridge, args[0]);
  double expr_value = EvaluateExpression(expression_array);

  // If there was a second argument, assume it's the oncalculate callback.
  if (arg_count > 1 && NPVARIANT_IS_OBJECT(args[1])) {
    // The ObjectType ctor bumps the ref count of the callback function.
    c_salt::ObjectType function_obj(NPVARIANT_TO_OBJECT(args[1]));
    // Pack the value of the expression into the first arg of the callback
    // function, and invoke it.
    NPVariant argv;
    NPVariant result;
    DOUBLE_TO_NPVARIANT(expr_value, argv);
    NULL_TO_NPVARIANT(result);
    NPN_InvokeDefault(bridge->npp(),
                      function_obj.object_value(),
                      &argv,
                      1,
                      &result);
  }

  return true;
}
```

Tutorial: Adding Methods

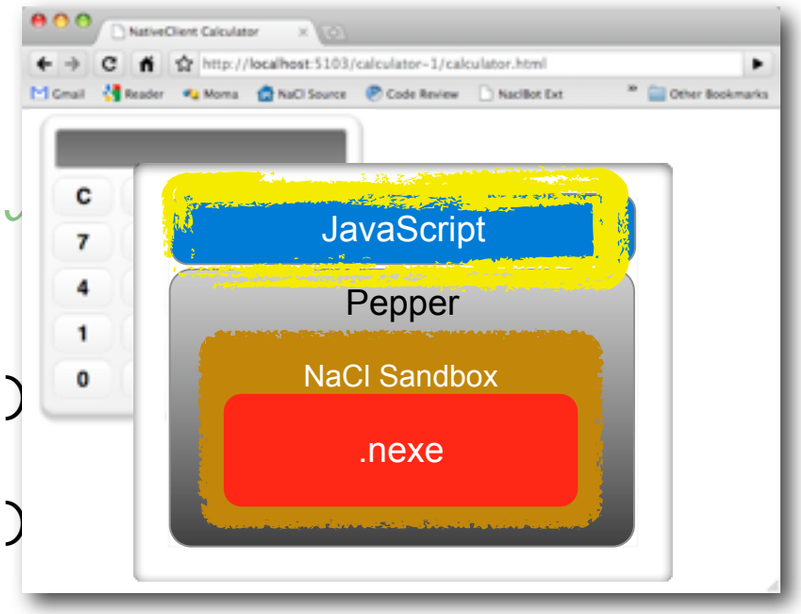
JavaScript

```
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
  } else {
    document.getElementById('formula').value = result;
  }
}
```

Tutorial: Adding Methods

JavaScript

```
  // into an array that gets passed to the calculator mod
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module()
  try {
    calculator_module.calculate(postfixExpr, onCalculate)
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
  } else {
    document.getElementById('formula').value = result;
  }
}
```



Tutorial: Adding Methods

JavaScript

```
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err) {
    onCalculate('42, I think');
  }
}

/**
 * Handler for the event when the calculation is complete.
 */
function onCalculate(result, opt_error) {
  if (opt_error) {
    alert(opt_error);
  } else {
    document.getElementById('formula').value = result;
  }
}
```

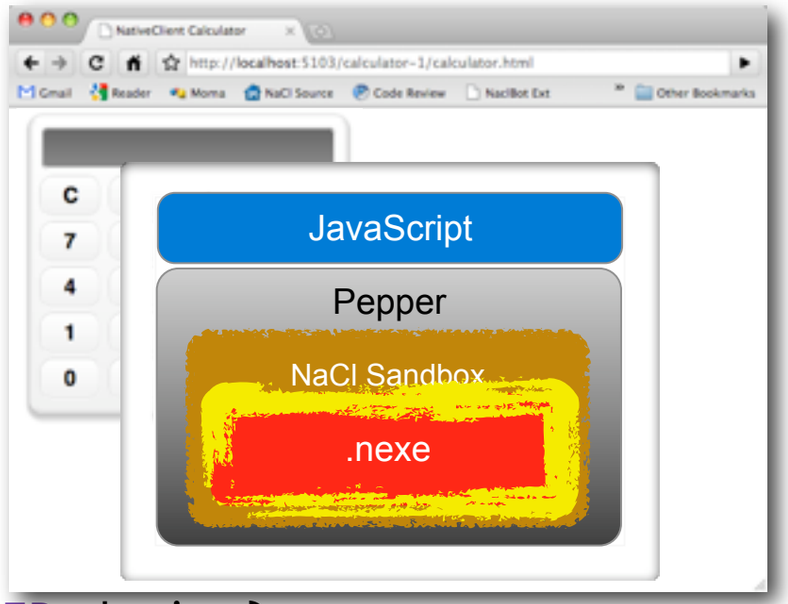
Calculator Tutorial Step 2

Tutorial: Adding Sound

C++

```
#include "c_salt/object_type.h"
#include "c_salt/scripting_bridge.h"
#include "c_salt/string_type.h"

#include <limits>

extern NPDevice* NPN_AcquireDevice(NPP instance, NPDeviceID device);
extern const int16_t click_sound_samples[];
extern const int kClickSampleCount;

namespace c_salt {

// The required factory method.
Module* Module::CreateModule() {
  return new calculator::Calculator();
}

}  // namespace c_salt

// Audio device call-back wrapper.
void AudioCallback(NPDeviceContextAudio *context) {
  if (!context)
    return;
  calculator::Calculator* calculator =
      static_cast<calculator::Calculator*>(context->config.userData);
```

Tutorial: Adding Sound



C++

```
#include "c_salt/object_type.h"
#include "c_salt/scripting_bridge.h"
#include "c_salt/string_type.h"

#include <limits>

extern NPDevice* NPN_AcquireDevice(NPP instance, NPDeviceID device);
extern const int16_t click_sound_samples[];
extern const int kClickSampleCount;

namespace c_salt {

// The required factory method.
Module* Module::CreateModule() {
  return new calculator::Calculator();
}

}  // namespace c_salt

// Audio device call-back wrapper.
void AudioCallback(NPDeviceContextAudio *context) {
  if (!context)
    return;
  calculator::Calculator* calculator =
      static_cast<calculator::Calculator*>(context->config.userData);
```

Tutorial: Adding Sound

C++

```
#include "c_salt/object_type.h"
#include "c_salt/scripting_bridge.h"
#include "c_salt/string_type.h"

#include <limits>

extern NPDevice* NPN_AcquireDevice(NPP instance, NPDeviceID device);
extern const int16_t click_sound_samples[];
extern const int kClickSampleCount;

namespace c_salt {

// The required factory method.
Module* Module::CreateModule() {
  return new calculator::Calculator();
}

}  // namespace c_salt

// Audio device call-back wrapper.
void AudioCallback(NPDeviceContextAudio *context) {
  if (!context)
    return;
  calculator::Calculator* calculator =
      static_cast<calculator::Calculator*>(context->config.userData);
```

Tutorial: Adding Sound

C++

```
    calculate_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
    bridge->AddMethodNamed("calculate", calculate_callback_);
    click_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Click);
    bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
    get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
        this, &Calculator::GetButtonSound);
    set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
        this, &Calculator::SetButtonSound);
    bridge->AddPropertyNamed("buttonSound",
                             get_button_sound_,
                             set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
    if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
        return false;

    c_salt::Type::TypeArray* expression_array =
```

Tutorial: Adding Sound

C++

```
    calculate_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
    bridge->AddMethodNamed("calculate", calculate_callback_);
    click_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Click);
    bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
    get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
        this, &Calculator::GetButtonSound);
    set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
        this, &Calculator::SetButtonSound);
    bridge->AddPropertyNamed("buttonSound",
                             get_button_sound_,
                             set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
    if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
      return false;

    c_salt::Type::TypeArray* expression_array =
```

Tutorial: Adding Sound

C++

```
    calculate_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
    bridge->AddMethodNamed("calculate", calculate_callback_);
    click_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Click);
    bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
    get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
        this, &Calculator::GetButtonSound);
    set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
        this, &Calculator::SetButtonSound);
    bridge->AddPropertyNamed("buttonSound",
                             get_button_sound_,
                             set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
    if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
        return false;

    c_salt::Type::TypeArray* expression_array =
```

Tutorial: Adding Sound

C++

```
    calculate_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
    bridge->AddMethodNamed("calculate", calculate_callback_);
    click_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Click);
    bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
    get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
        this, &Calculator::GetButtonSound);
    set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
        this, &Calculator::SetButtonSound);
    bridge->AddPropertyNamed("buttonSound",
                             get_button_sound_,
                             set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
    if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
      return false;

    c_salt::Type::TypeArray* expression_array =
```

Tutorial: Adding Sound

C++

```
    calculate_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
    bridge->AddMethodNamed("calculate", calculate_callback_);
    click_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Click);
    bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
    get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
        this, &Calculator::GetButtonSound);
    set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
        this, &Calculator::SetButtonSound);
    bridge->AddPropertyNamed("buttonSound",
                             get_button_sound_,
                             set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
    if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
      return false;

    c_salt::Type::TypeArray* expression_array =
```

Tutorial: Adding Sound

C++

```
  calculate_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
  bridge->AddMethodNamed("calculate", calculate_callback_);
  click_callback_ =
      new c_salt::MethodCallback<Calculator>(this, &Calculator::Click);
  bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
  get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
      this, &Calculator::GetButtonSound);
  set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
      this, &Calculator::SetButtonSound);
  bridge->AddPropertyNamed("buttonSound",
                           get_button_sound_,
                           set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
  if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
    return false;

  c_salt::Type::TypeArray* expression_array =
```

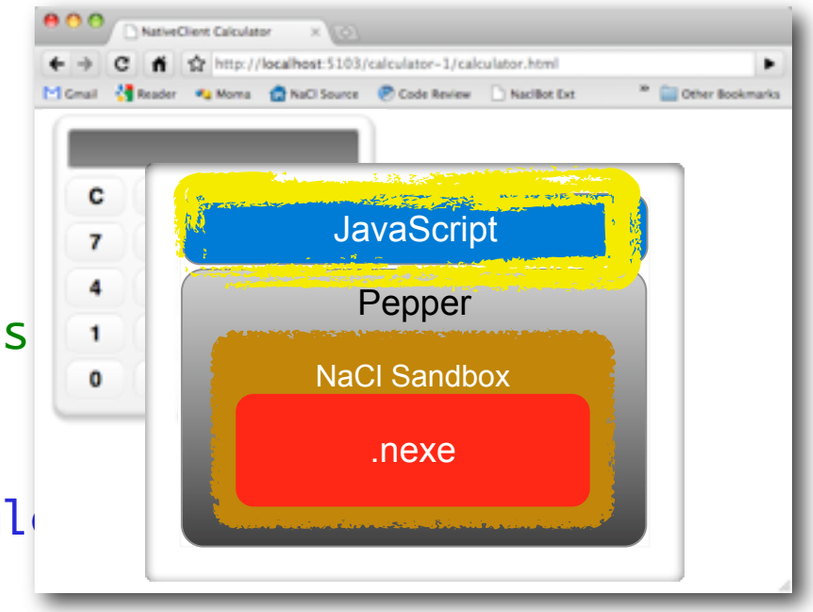
Tutorial: Adding Sound

C++

```
/**
 * This function is called when the calculator module is loaded.
 */
function moduleDidLoad() {
  var module = document.getElementById('calculator_module');
  calculator.application.moduleDidLoad(module);
  var calculator_module = calculator.application.module();
  calculator_module.buttonSound = true;
}

/**
 * Play the click sound if sounds are turned on.
 */
function click() {
  var calculator_module = calculator.application.module();
  if (calculator_module && calculator_module.buttonSound) {
    calculator_module.click();
  }
}
```

Tutorial: Adding Sound

C++




```
/**
 * This function is called when the calculator module is
 */
function moduleDidLoad() {
  var module = document.getElementById('calculator_modul
  calculator.application.moduleDidLoad(module);
  var calculator_module = calculator.application.module();
  calculator_module.buttonSound = true;
}

/**
 * Play the click sound if sounds are turned on.
 */
function click() {
  var calculator_module = calculator.application.module();
  if (calculator_module && calculator_module.buttonSound) {
    calculator_module.click();
  }
}
```

Tutorial: Adding Sound

C++

```
/**
 * This function is called when the calculator module is loaded.
 */
function moduleDidLoad() {
  var module = document.getElementById('calculator_module');
  calculator.application.moduleDidLoad(module);
  var calculator_module = calculator.application.module();
  calculator_module.buttonSound = true;
}

/**
 * Play the click sound if sounds are turned on.
 */
function click() {
  var calculator_module = calculator.application.module();
  if (calculator_module && calculator_module.buttonSound) {
    calculator_module.click();
  }
}
```

Tutorial: Adding Sound

C++

```
/**
 * This function is called when the calculator module is loaded.
 */
function moduleDidLoad() {
  var module = document.getElementById('calculator_module');
  calculator.application.moduleDidLoad(module);
  var calculator_module = calculator.application.module();
  calculator_module.buttonSound = true;
}

/**
 * Play the click sound if sounds are turned on.
 */
function click() {
  var calculator_module = calculator.application.module();
  if (calculator_module && calculator_module.buttonSound) {
    calculator_module.click();
  }
}
```

Tutorial: Adding Sound

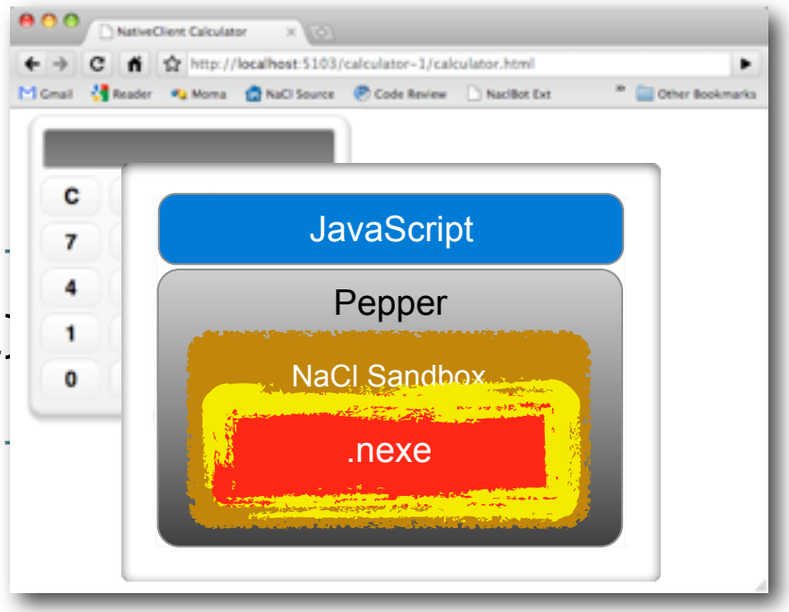
C++

```
    calculate_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
    bridge->AddMethodNamed("calculate", calculate_callback_);
    click_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Click);
    bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
    get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
        this, &Calculator::GetButtonSound);
    set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
        this, &Calculator::SetButtonSound);
    bridge->AddPropertyNamed("buttonSound",
                             get_button_sound_,
                             set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
    if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
      return false;

    c_salt::Type::TypeArray* expression_array =
```

Tutorial: Adding Sound

C++

```
    calculate_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calcu
    bridge->AddMethodNamed("calculate", calculate_callback_
    click_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calcu
    bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
    get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
        this, &Calculator::GetButtonSound);
    set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
        this, &Calculator::SetButtonSound);
    bridge->AddPropertyNamed("buttonSound",
                             get_button_sound_,
                             set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
    if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
      return false;

    c_salt::Type::TypeArray* expression_array =
```



Tutorial: Adding Sound

C++

```
    calculate_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Calculate);
    bridge->AddMethodNamed("calculate", calculate_callback_);
    click_callback_ =
        new c_salt::MethodCallback<Calculator>(this, &Calculator::Click);
    bridge->AddMethodNamed("click", click_callback_);
}

void Calculator::InitializeProperties(c_salt::ScriptingBridge* bridge) {
    get_button_sound_ = new c_salt::PropertyAccessorCallback<Calculator>(
        this, &Calculator::GetButtonSound);
    set_button_sound_ = new c_salt::PropertyMutatorCallback<Calculator>(
        this, &Calculator::SetButtonSound);
    bridge->AddPropertyNamed("buttonSound",
                             get_button_sound_,
                             set_button_sound_);
}

bool Calculator::Calculate(c_salt::ScriptingBridge* bridge,
                           const NPVariant* args,
                           uint32_t arg_count,
                           NPVariant* return_value) {
    if (arg_count < 1 || !NPVARIANT_IS_OBJECT(args[0]))
      return false;

    c_salt::Type::TypeArray* expression_array =
```

Tutorial: Adding Sound

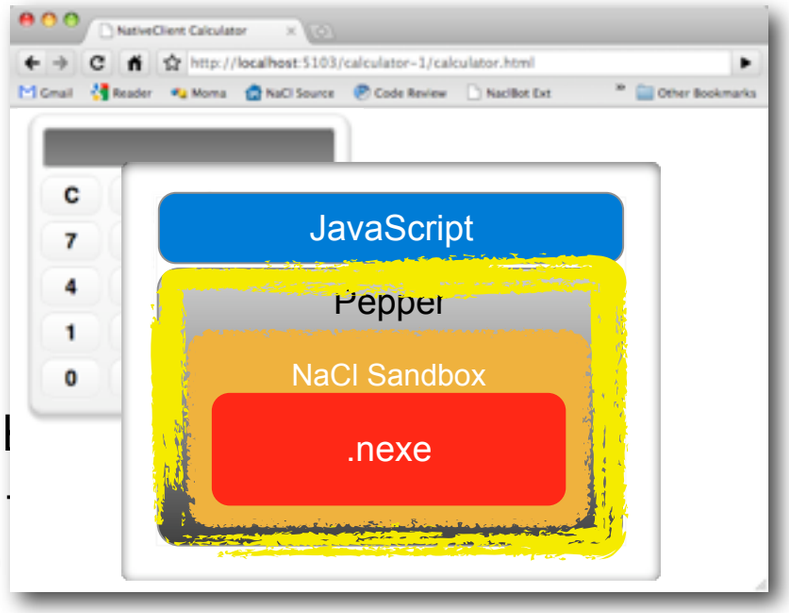
C++

```
  return true;
}

bool Calculator::GetButtonSound(c_salt::ScriptingBridge* bridge,
                                NPVariant* return_value) {
  BOOLEAN_TO_NPVARIANT(play_button_sound_, *return_value);
  return true;
}

bool Calculator::SetButtonSound(c_salt::ScriptingBridge* bridge,
                                const NPVariant* value) {
  if (!value)
    return false;
  c_salt::Type* set_sound = c_salt::Type::CreateFromNPVariant(*value);
  play_button_sound_ = set_sound ? set_sound->bool_value() : false;
  delete set_sound;
  return true;
}

double Calculator::EvaluateExpression(
    const c_salt::Type::TypeArray* expression) {
  if (expression->size() == 0)
    return 0.0;
  // This is a pretty simple-minded expression evaulator.  It assumes that the
  // input vector represents a correct postfix expression and does basically
```

Tutorial: Adding Sound

C++

```
  return true;
}

bool Calculator::GetButtonSound(c_salt::ScriptingBridge* b
                                NPVariant* return_value) {
  BOOLEAN_TO_NPVARIANT(play_button_sound_, *return_value);
  return true;
}

bool Calculator::SetButtonSound(c_salt::ScriptingBridge* bridge,
                                const NPVariant* value) {
  if (!value)
    return false;
  c_salt::Type* set_sound = c_salt::Type::CreateFromNPVariant(*value);
  play_button_sound_ = set_sound ? set_sound->bool_value() : false;
  delete set_sound;
  return true;
}

double Calculator::EvaluateExpression(
    const c_salt::Type::TypeArray* expression) {
  if (expression->size() == 0)
    return 0.0;
  // This is a pretty simple-minded expression evaulator.  It assumes that the
  // input vector represents a correct postfix expression and does basically
```



Tutorial: Adding Sound

C++

```
  return true;
}

bool Calculator::GetButtonSound(c_salt::ScriptingBridge* bridge,
                                NPVariant* return_value) {
  BOOLEAN_TO_NPVARIANT(play_button_sound_, *return_value);
  return true;
}

bool Calculator::SetButtonSound(c_salt::ScriptingBridge* bridge,
                                const NPVariant* value) {
  if (!value)
    return false;
  c_salt::Type* set_sound = c_salt::Type::CreateFromNPVariant(*value);
  play_button_sound_ = set_sound ? set_sound->bool_value() : false;
  delete set_sound;
  return true;
}

double Calculator::EvaluateExpression(
    const c_salt::Type::TypeArray* expression) {
  if (expression->size() == 0)
    return 0.0;
  // This is a pretty simple-minded expression evaulator.  It assumes that the
  // input vector represents a correct postfix expression and does basically
```

Tutorial: Adding Sound

C++

```
  return true;
}

bool Calculator::GetButtonSound(c_salt::ScriptingBridge* bridge,
                                NPVariant* return_value) {
  BOOLEAN_TO_NPVARIANT(play_button_sound_, *return_value);
  return true;
}

bool Calculator::SetButtonSound(c_salt::ScriptingBridge* bridge,
                                const NPVariant* value) {
  if (!value)
    return false;
  c_salt::Type* set_sound = c_salt::Type::CreateFromNPVariant(*value);
  play_button_sound_ = set_sound ? set_sound->bool_value() : false;
  delete set_sound;
  return true;
}

double Calculator::EvaluateExpression(
    const c_salt::Type::TypeArray* expression) {
  if (expression->size() == 0)
    return 0.0;
  // This is a pretty simple-minded expression evaulator.  It assumes that the
  // input vector represents a correct postfix expression and does basically
```

Tutorial: Adding Sound

C++

```
  return true;
}

bool Calculator::GetButtonSound(c_salt::ScriptingBridge* bridge,
                                NPVariant* return_value) {
  BOOLEAN_TO_NPVARIANT(play_button_sound_, *return_value);
  return true;
}

bool Calculator::SetButtonSound(c_salt::ScriptingBridge* bridge,
                                const NPVariant* value) {
  if (!value)
    return false;
  c_salt::Type* set_sound = c_salt::Type::CreateFromNPVariant(*value);
  play_button_sound_ = set_sound ? set_sound->bool_value() : false;
  delete set_sound;
  return true;
}

double Calculator::EvaluateExpression(
    const c_salt::Type::TypeArray* expression) {
  if (expression->size() == 0)
    return 0.0;
  // This is a pretty simple-minded expression evaulator.  It assumes that the
  // input vector represents a correct postfix expression and does basically
```

Tutorial: Adding Sound

C++

```
  return true;
}

bool Calculator::GetButtonSound(c_salt::ScriptingBridge* bridge,
                                NPVariant* return_value) {
  BOOLEAN_TO_NPVARIANT(play_button_sound_, *return_value);
  return true;
}

bool Calculator::SetButtonSound(c_salt::ScriptingBridge* bridge,
                                const NPVariant* value) {
  if (!value)
    return false;
  c_salt::Type* set_sound = c_salt::Type::CreateFromNPVariant(*value);
  play_button_sound_ = set_sound ? set_sound->bool_value() : false;
  delete set_sound;
  return true;
}

double Calculator::EvaluateExpression(
    const c_salt::Type::TypeArray* expression) {
  if (expression->size() == 0)
    return 0.0;
  // This is a pretty simple-minded expression evaulator.  It assumes that the
  // input vector represents a correct postfix expression and does basically
```

Tutorial: Adding Sound

C++

```
void Calculator::SetWindow(NPP instance, int width, int height) {
  if (device_audio_)
    return;
  // Initialize the audio context.
  device_audio_ = NPN_AcquireDevice(instance, NPPepperAudioDevice);
  assert(device_audio_);
  memset(&context_audio_, 0, sizeof(context_audio_));
  NPDeviceContextAudioConfig cfg;
  cfg.sampleRate = 44100;
  cfg.sampleType = NPAudioSampleTypeInt16;
  cfg.outputChannelMap = NPAudioChannelStereo;
  cfg.inputChannelMap = NPAudioChannelNone;
  cfg.sampleFrameCount = 1024;
  cfg.startThread = 1;  // Start a thread for the audio producer.
  cfg.flags = 0;
  cfg.callback = &AudioCallback;
  cfg.userData = reinterpret_cast<void*>(this);
  NPError init_err = device_audio_->initializeContext(instance,
                                                      &cfg,
                                                      &context_audio_);
  assert(NPERR_NO_ERROR == init_err);
}

}  // namespace calculator
```

Tutorial: Adding Sound

C++

```
void Calculator::SetWindow(NPP instance, int width, int height) {
  if (device_audio_)
    return;
  // Initialize the audio context.
  device_audio_ = NPN_AcquireDevice(instance, NPPepperAudioDevice);
  assert(device_audio_);
  memset(&context_audio_, 0, sizeof(context_audio_));
  NPDeviceContextAudioConfig cfg;
  cfg.sampleRate = 44100;
  cfg.sampleType = NPAudioSampleTypeInt16;
  cfg.outputChannelMap = NPAudioChannelStereo;
  cfg.inputChannelMap = NPAudioChannelNone;
  cfg.sampleFrameCount = 1024;
  cfg.startThread = 1;  // Start a thread for the audio producer.
  cfg.flags = 0;
  cfg.callback = &AudioCallback;
  cfg.userData = reinterpret_cast<void*>(this);
  NPError init_err = device_audio_->initializeContext(instance,
                                                      &cfg,
                                                      &context_audio_);
  assert(NPERR_NO_ERROR == init_err);
}

}  // namespace calculator
```

Tutorial: Adding Sound

C++

```
void Calculator::SetWindow(NPP instance, int width, int height) {
  if (device_audio_)
    return;
  // Initialize the audio context.
  device_audio_ = NPN_AcquireDevice(instance, NPPepperAudioDevice);
  assert(device_audio_);
  memset(&context_audio_, 0, sizeof(context_audio_));
  NPDeviceContextAudioConfig cfg;
  cfg.sampleRate = 44100;
  cfg.sampleType = NPAudioSampleTypeInt16;
  cfg.outputChannelMap = NPAudioChannelStereo;
  cfg.inputChannelMap = NPAudioChannelNone;
  cfg.sampleFrameCount = 1024;
  cfg.startThread = 1;  // Start a thread for the audio producer.
  cfg.flags = 0;
  cfg.callback = &AudioCallback;
  cfg.userData = reinterpret_cast<void*>(this);
  NPError init_err = device_audio_->initializeContext(instance,
                                                      &cfg,
                                                      &context_audio_);
  assert(NPERR_NO_ERROR == init_err);
}

}  // namespace calculator
```

Tutorial: Adding Sound

C++

```
void Calculator::SetWindow(NPP instance, int width, int height) {
  if (device_audio_)
    return;
  // Initialize the audio context.
  device_audio_ = NPN_AcquireDevice(instance, NPPepperAudioDevice);
  assert(device_audio_);
  memset(&context_audio_, 0, sizeof(context_audio_));
  NPDeviceContextAudioConfig cfg;
  cfg.sampleRate = 44100;
  cfg.sampleType = NPAudioSampleTypeInt16;
  cfg.outputChannelMap = NPAudioChannelStereo;
  cfg.inputChannelMap = NPAudioChannelNone;
  cfg.sampleFrameCount = 1024;
  cfg.startThread = 1;  // Start a thread for the audio producer.
  cfg.flags = 0;
  cfg.callback = &AudioCallback;
  cfg.userData = reinterpret_cast<void*>(this);
  NPError init_err = device_audio_->initializeContext(instance,
                                                      &cfg,
                                                      &context_audio_);
  assert(NPERR_NO_ERROR == init_err);
}

}  // namespace calculator
```

Tutorial: Adding Sound

C++

```
  return new calculator::Calculator();
}

}  // namespace c_salt

// Audio device call-back wrapper.
void AudioCallback(NPDeviceContextAudio *context) {
  if (!context)
    return;
  calculator::Calculator* calculator =
      static_cast<calculator::Calculator*>(context->config.userData);
  if (calculator) {
    calculator->SynthesizeButtonClick(context);
  }
}

namespace calculator {

Calculator::Calculator()
    : calculate_callback_(NULL),
      click_callback_(NULL),
      get_button_sound_(NULL),
      set_button_sound_(NULL),
      play_button_sound_(false),
      click_pending_(false),
```

Tutorial: Adding Sound

C++

```
  return new calculator::Calculator();
}

}  // namespace c_salt

// Audio device call-back wrapper.
void AudioCallback(NPDeviceContextAudio *context) {
  if (!context)
    return;
  calculator::Calculator* calculator =
      static_cast<calculator::Calculator*>(context->config.userData);
  if (calculator) {
    calculator->SynthesizeButtonClick(context);
  }
}

namespace calculator {

Calculator::Calculator()
    : calculate_callback_(NULL),
      click_callback_(NULL),
      get_button_sound_(NULL),
      set_button_sound_(NULL),
      play_button_sound_(false),
      click_pending_(false),
```

Tutorial: Adding Sound

C++

```
  return new calculator::Calculator();
}

}  // namespace c_salt

// Audio device call-back wrapper.
void AudioCallback(NPDeviceContextAudio *context) {
  if (!context)
    return;
  calculator::Calculator* calculator =
      static_cast<calculator::Calculator*>(context->config.userData);
  if (calculator) {
    calculator->SynthesizeButtonClick(context);
  }
}

namespace calculator {

Calculator::Calculator()
    : calculate_callback_(NULL),
      click_callback_(NULL),
      get_button_sound_(NULL),
      set_button_sound_(NULL),
      play_button_sound_(false),
      click_pending_(false),
```

Tutorial: Adding Sound

C++

```
}

void Calculator::SynthesizeButtonClick(NPDeviceContextAudio *context) {
  const size_t sample_count  = context->config.sampleFrameCount;
  const size_t channel_count = context->config.outputChannelMap;
  const size_t sample_copy_count = sample_count * channel_count;
  int16* buf = reinterpret_cast<int16*>(context->outBuffer);
  memset(buf, 0, sample_copy_count * sizeof(int16));
  if (click_pending_) {
    size_t remaining_samples = kClickSampleCount - current_sample_index_;
    size_t copy_count = std::min(sample_copy_count, remaining_samples);
    memcpy(buf,
           &click_sound_samples[current_sample_index_],
           copy_count * sizeof(int16));
    current_sample_index_ += sample_copy_count;
    if (current_sample_index_ > kClickSampleCount) {
      current_sample_index_ = 0;
      // Indicate that the sound has finished playing.
      click_pending_ = false;
    }
  }
}

void Calculator::SetWindow(NPP instance, int width, int height) {
  if (device_audio_)
```

Calculator Tutorial Step 3

Coming Attractions

- Full debugger support
  - GDB
  - Visual Studio
- IDE plugins
  - Eclipse
  - Visual Studio
  - Xcode
- Easy to use interop library
- P-NaCl (portable NaCl)

How To Contribute

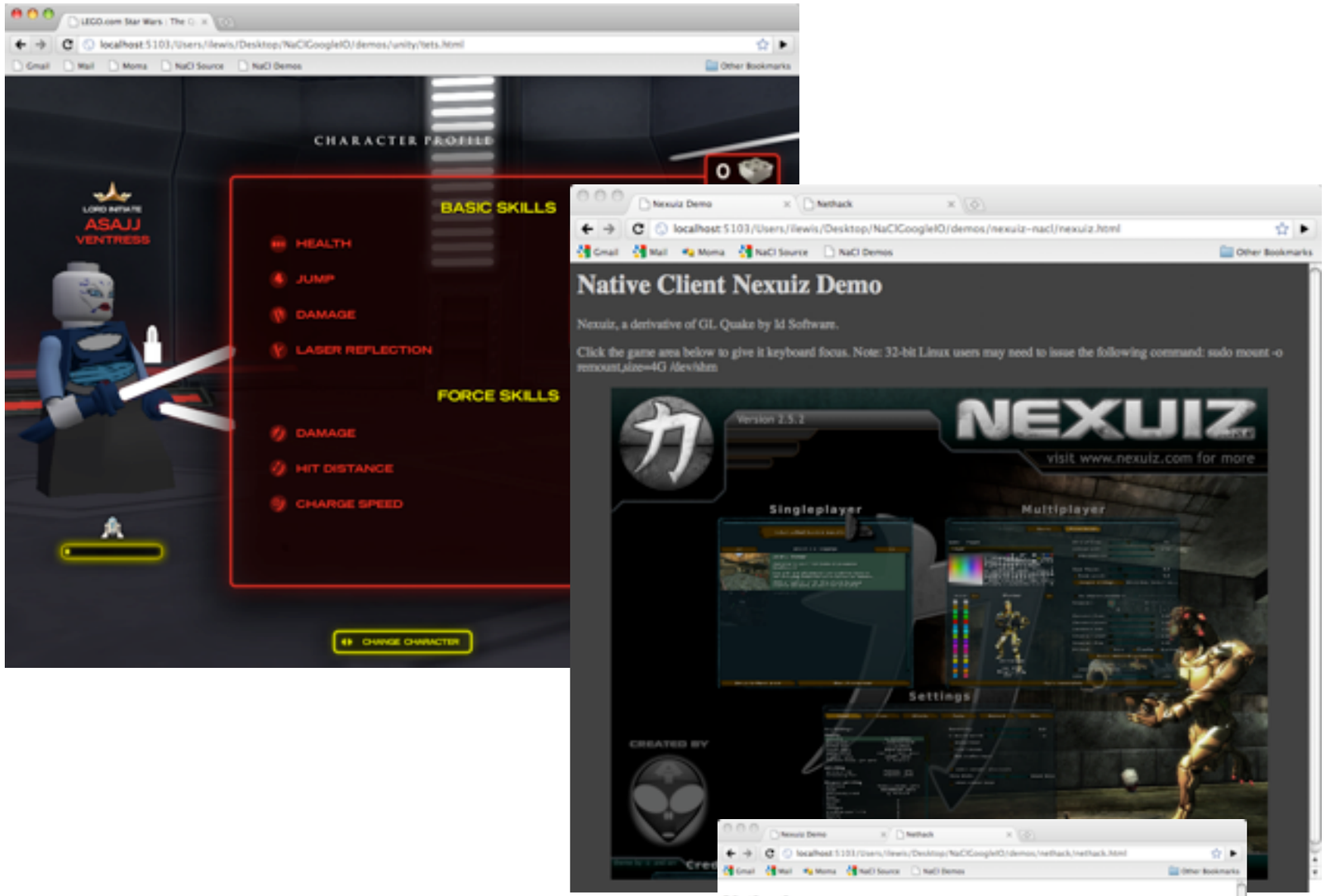
nativeclient.googlecode.com

nativeclient-sdk.googlecode.com

native-client-discuss@google.com

Want More?

- Unity
- Nexuiz
- Text-to-speech
- VNC client
- NetHack



Come see us in the developer sandbox!

Q&A

View live notes and ask questions about this session on Google Wave:

http://bit.ly/cgOGiI