Writing Web Apps in C++?

Eric Bidelman, Google
COSCUP / GNOME.Asia - Taipei, Taiwan
August 14, 2010

Agenda

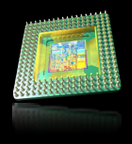
- Overview of Native Client
- SDK
- Calculator tutorial
- Demos

Native Client

Native Client

(NaCl)

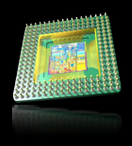
The Vision



NaCl

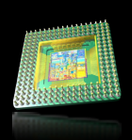
The Vision

NaCl



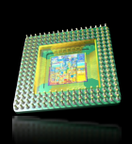
NaCl

"Write once, run anywhere"



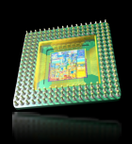
NaCl

"Write once, run anywhere"

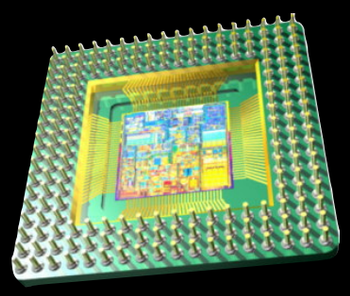


NaCl

+ + =



NaCl

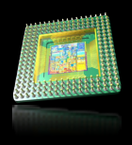


+

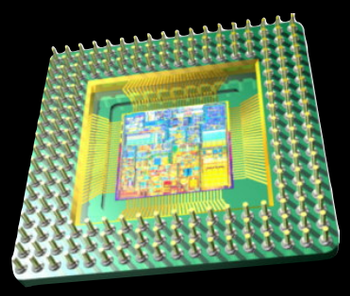
+

=

x86-32
x86-64
ARM



NaCl

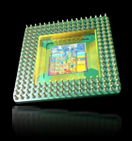


x86-32
x86-64
ARM

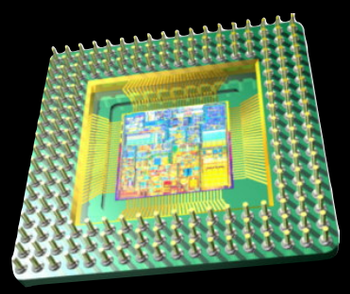
+

+

=

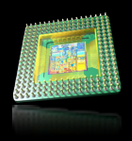


NaCl

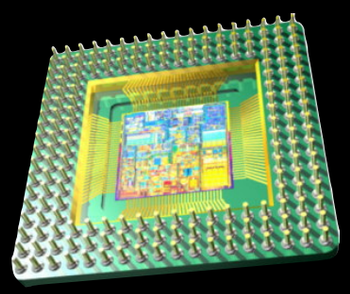


x86-32
x86-64
ARM

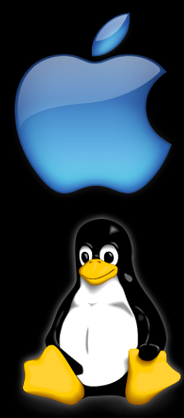
=

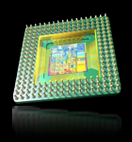


NaCl



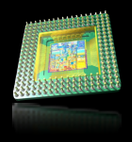
x86-32
x86-64
ARM





NaCl

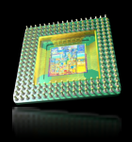

LEGO
STAR WARS
THE
QUEST FOR R2-D2
CLICK TO CONTINUE
DEVELOPED BY
THREE MELONS



NaCl

What is NaCl?

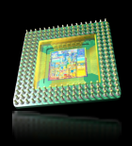
"Sandboxing technology for safe execution of platform-independent untrusted native code in a web browser. It allows web applications that are compute-intensive and/or interactive to leverage the resources of a client's machine and avoid costly network access while running in a secure environment with restricted access to the host."



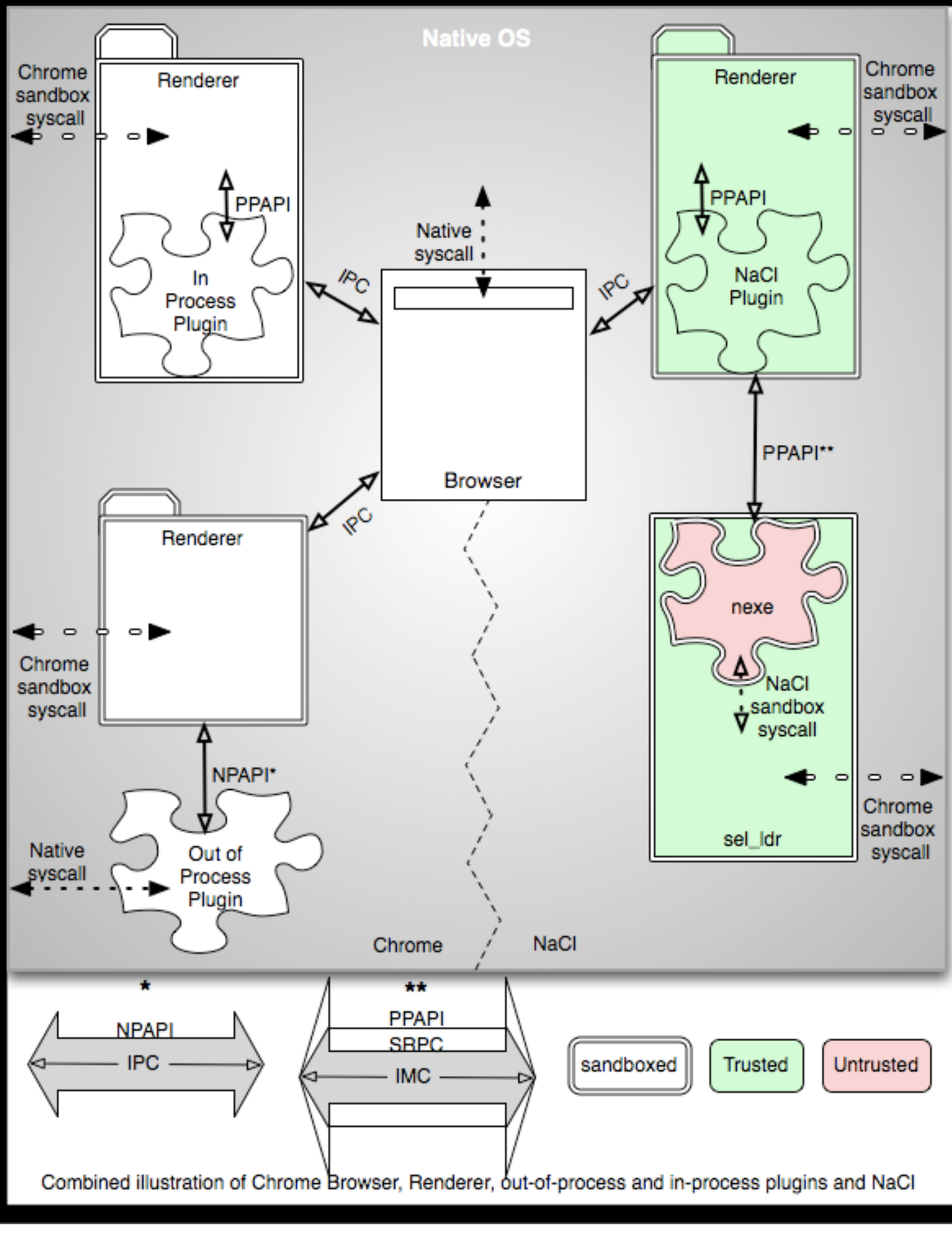
NaCl

What is NaCl?

"Sandboxing technology for safe execution of platform-independent untrusted native code in a web browser. It allows web applications that are compute-intensive and/or interactive to leverage the resources of a client's machine and avoid costly network access while running in a secure environment with restricted access to the host."



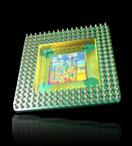
NaCl



Combined illustration of Chrome Browser, Renderer, out-of-process and in-process plugins and NaCl

What is it good for?

- Porting desktop applications to the web
  - No install
  - Native performance
  - e.g. games, media, large data analysis, visualizations
- 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



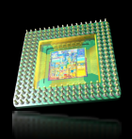
NaCl

Confirm Installation

This webpage wants to install software from MightNotBeEvil, Inc.

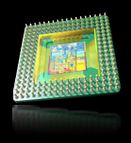
I know what you're thinking. Is this software safe? Well, to tell you the truth, we're not really sure ourselves. But being as this is a native plugin, most dangerous kind of code in the world, and could root your system without you even knowing it, you've got to ask yourself one question: Do I feel lucky? Well, do ya, punk?

OK Cancel



NaCl

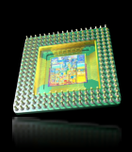
The NaCl SDK



NaCl

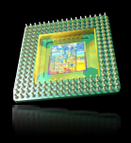
SDK Goodies

- C/C++ toolchain for creating NaCl executables (.nexe)
- Standard GNU libraries
- Customized versions of gcc, gdb, binutils
- “Pepper”
  - NPAPI-like interface
  - Audio / video
  - OpenGL ES 2.0
- Javascript interop layer (MVC): c_salt
- Sample code + build scripts



NaCl

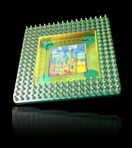
Calculator Tutorial



NaCl

Loading a .nexe

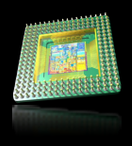
```
<div id="nacl_container"></div>

<script>









</script>
```



NaCl

Loading a .nexe

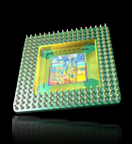
```
<div id="nacl_container"></div>

<script>


  var url = 'http://example.com/path/to/myModule.nexe';




</script>
```



NaCl

Loading a .nexe

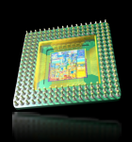
```
<div id="nacl_container"></div>

<script>


  var url = 'http://example.com/path/to/myModule.nexe';
  document.getElementById('nacl_container').innerHTML =
      ['<embed id="myModule" width="0" height="0" ',
       'src="', url, '" type="pepper-application/myModule" ',
       'onload="onModuleLoad();" />'].join('');




</script>
```



NaCl

Loading a .nexe

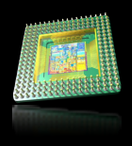
```
<div id="nacl_container"></div>

<script>
  function onModuleLoad() { alert('NaCl Module Loaded!'); }

  var url = 'http://example.com/path/to/myModule.nexe';
  document.getElementById('nacl_container').innerHTML =
      ['<embed id="myModule" width="0" height="0" ',
       'src="', url, '" type="pepper-application/myModule" ',
       'onload="onModuleLoad();" />'].join('');




</script>
```



NaCl

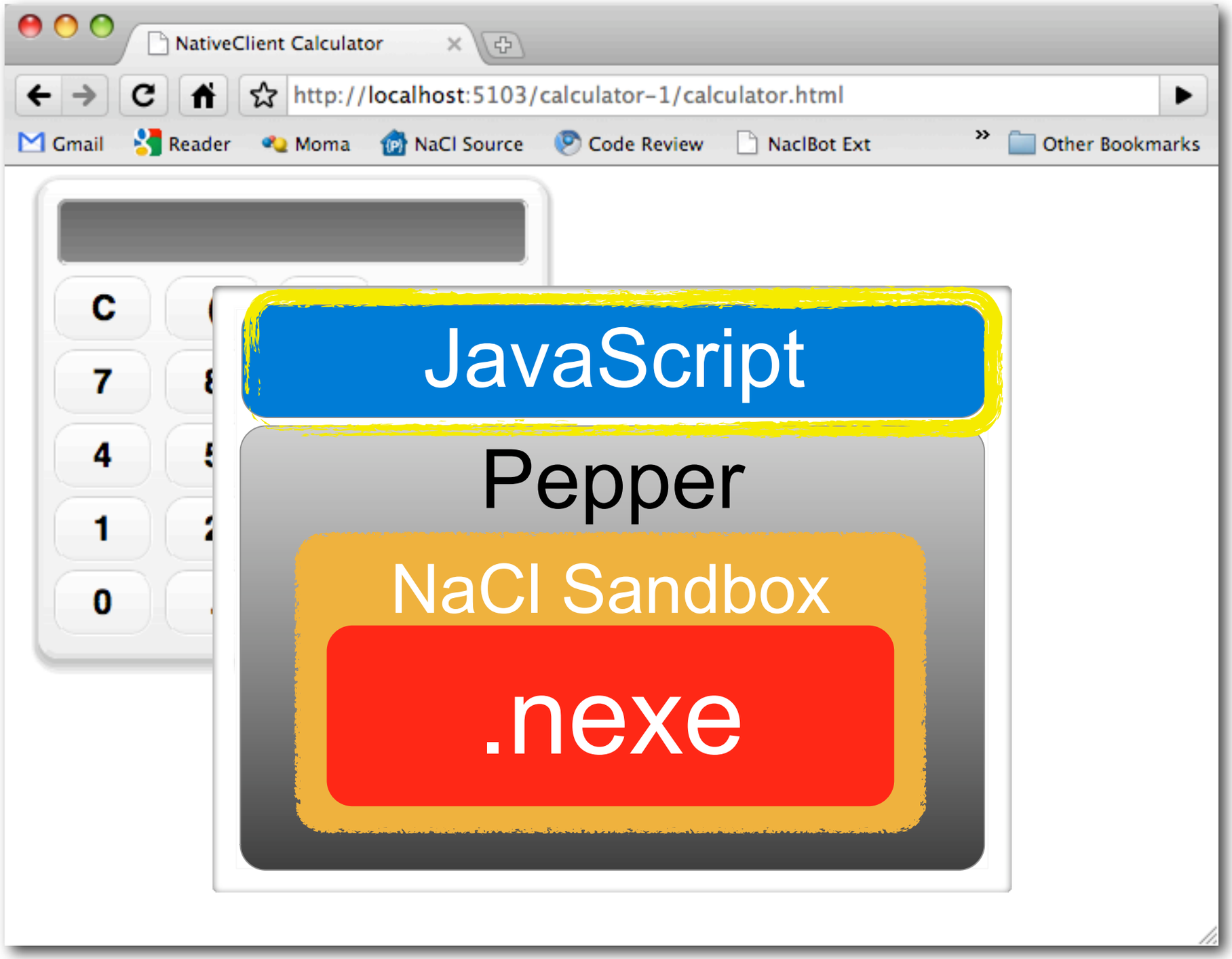
Loading a .nexe

```
<div id="nacl_container"></div>

<script>
  function onModuleLoad() { alert('NaCl Module Loaded!'); }

  var url = 'http://example.com/path/to/myModule.nexe';
  document.getElementById('nacl_container').innerHTML =
      ['<embed id="myModule" width="0" height="0" ',
       'src="', url, '" type="pepper-application/myModule" ',
       'onload="onModuleLoad();" />'].join('');

  // Next 2 lines are temporary due to:
  // http://code.google.com/p/nativeclient/issues/detail?id=500
  var nexes = 'x86-32: ../x86_32/myModule.nexe\n' +
              'x86-64: ../x86_64/myModule.nexe\n' +
              'ARM: ../arm/myModule.nexe';

  document.getElementById('myModule').nexes = nexes;
</script>
```

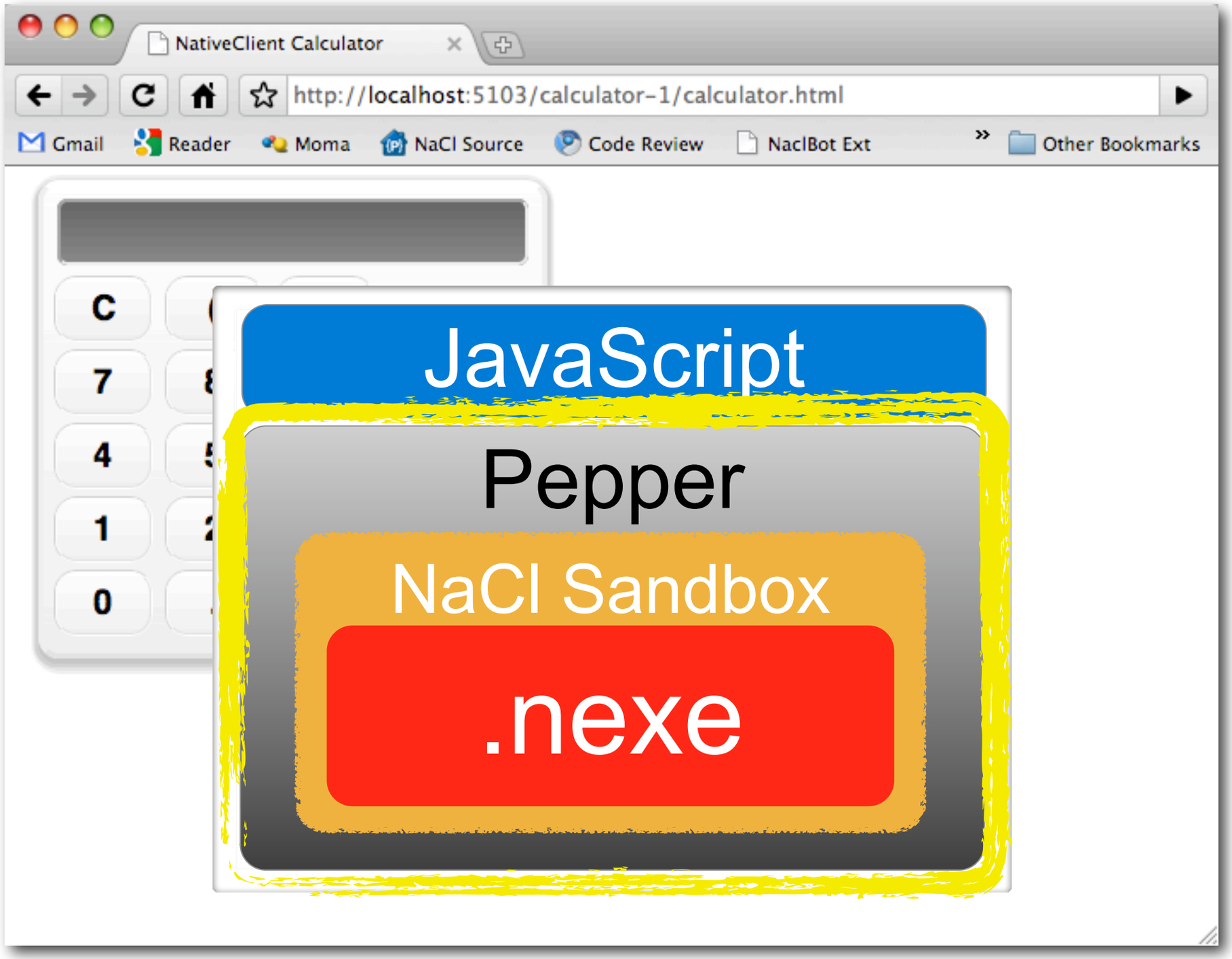


NaCl

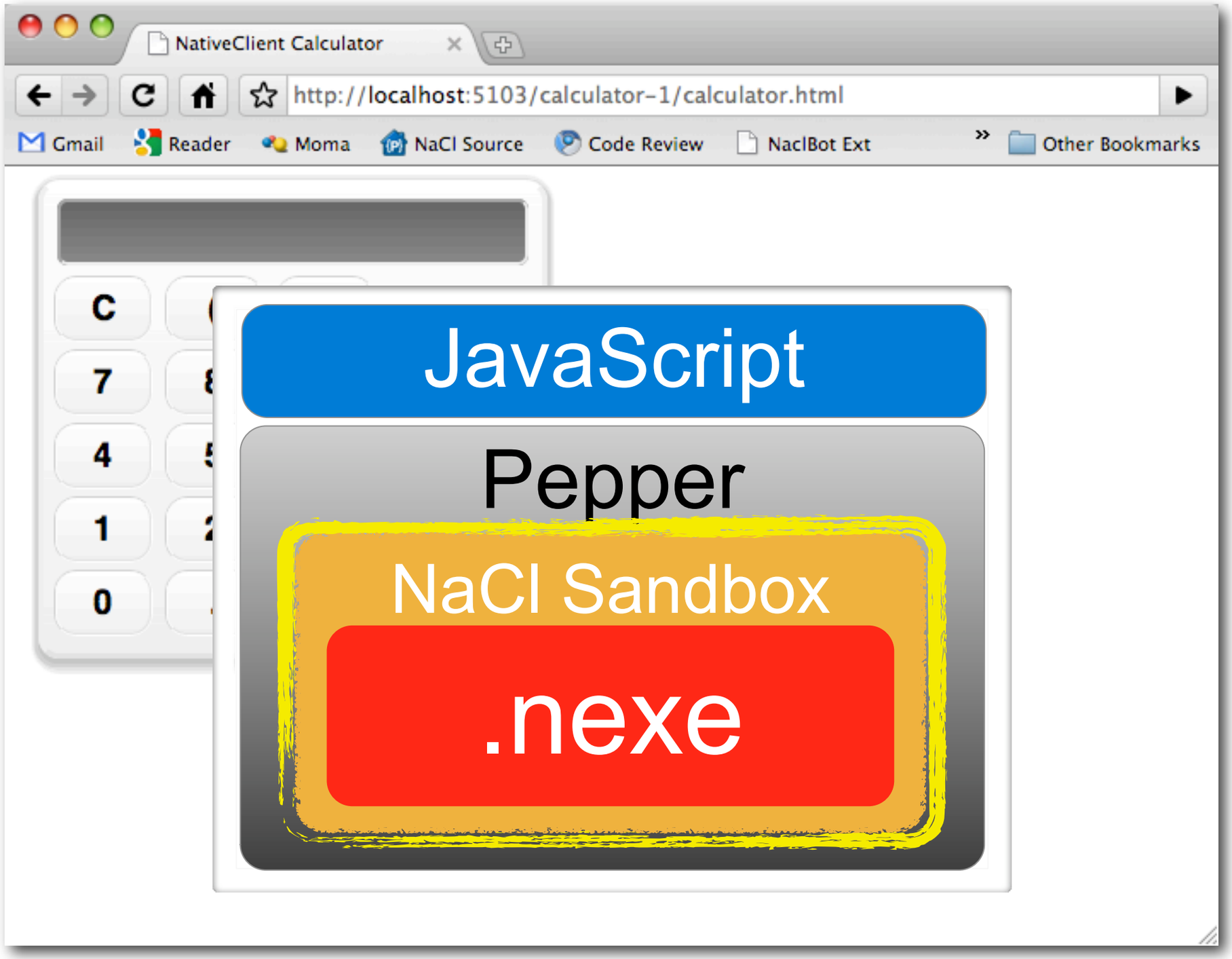
Anatomy of a NaCl Application



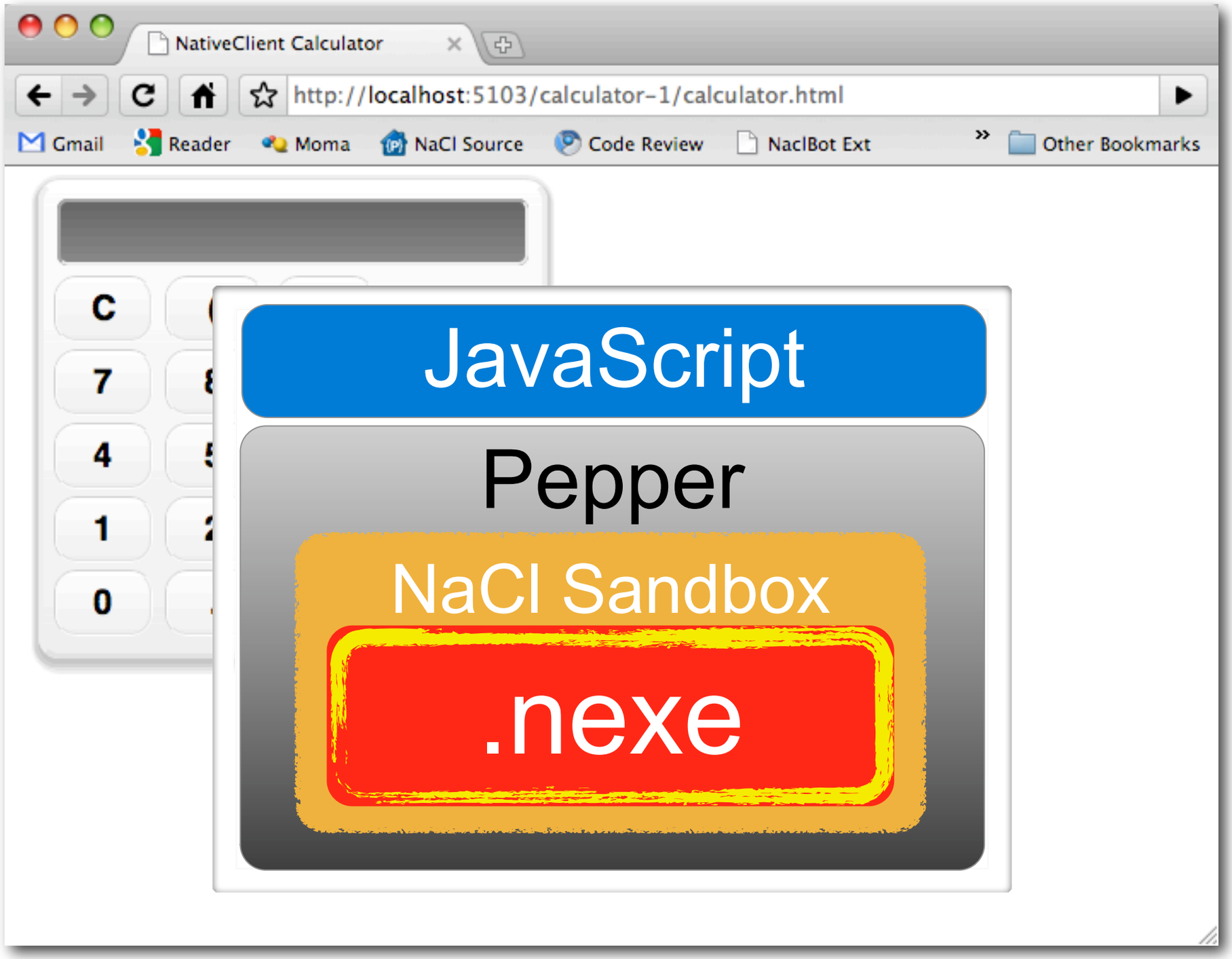
Anatomy of a NaCl Application



Anatomy of a NaCl Application



Anatomy of a NaCl Application



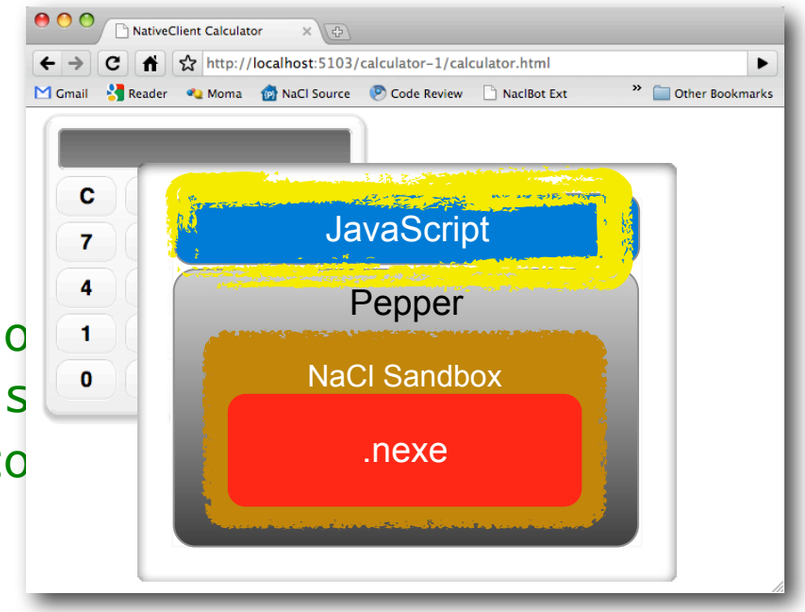
Tutorial: Adding UI

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);
```

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;
  }
}
```

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

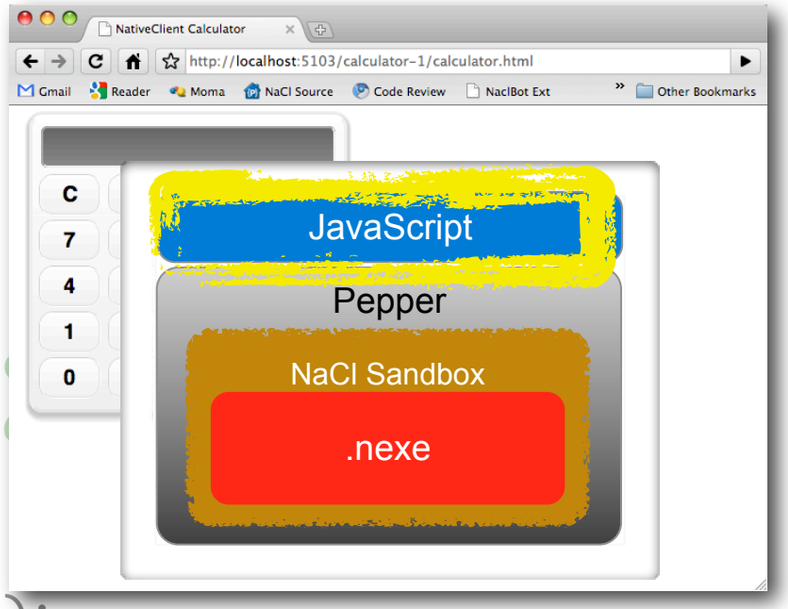
JavaScript

```
 */
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

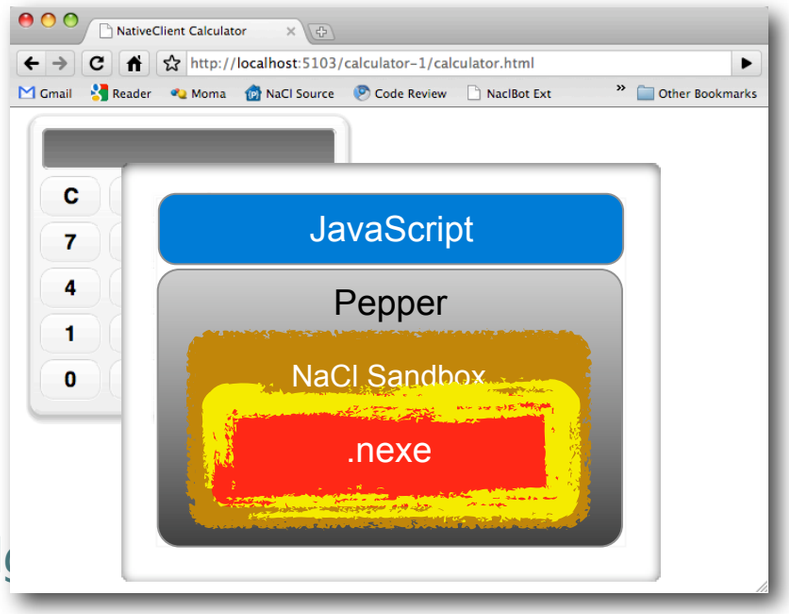
C++

```
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::ScriptingBridg
  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);
  }
```

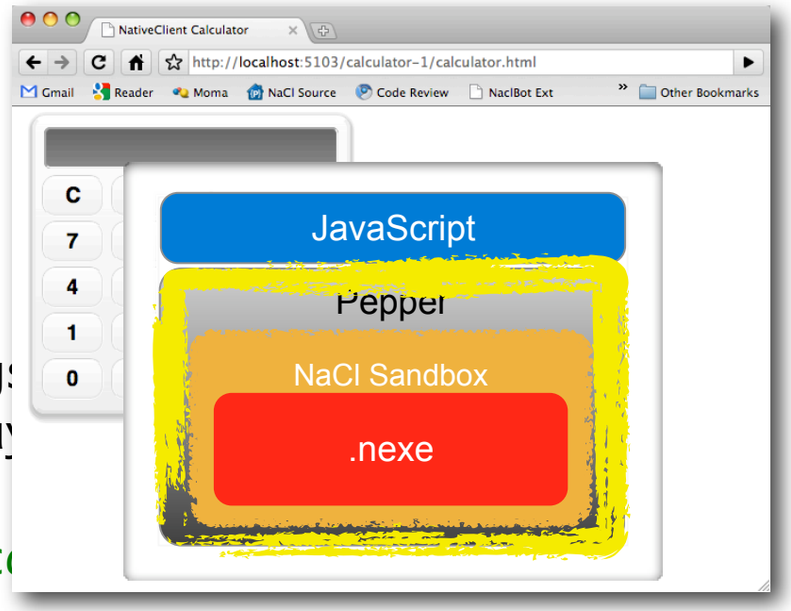
Tutorial: Adding Methods

C++

```
  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;
}

double CalculatorEvaluateExpression(
```

Tutorial: Adding Methods

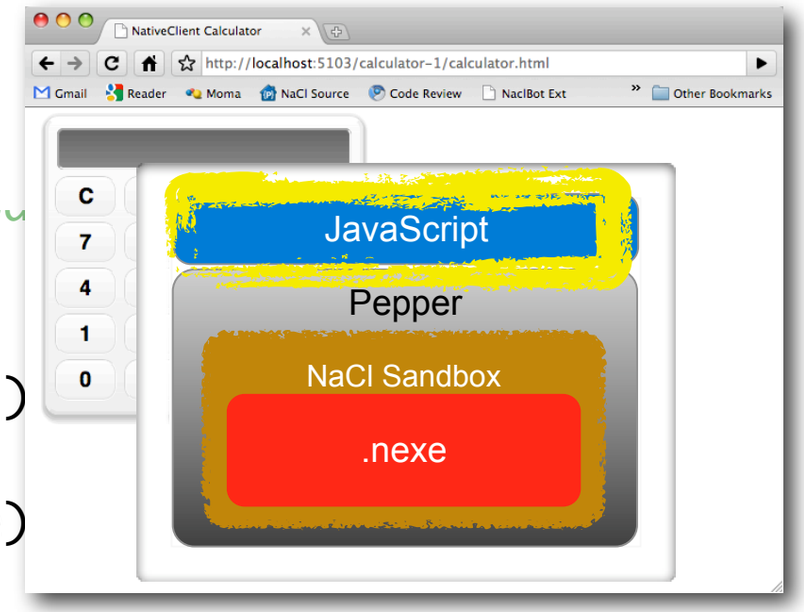
JavaScript

```
  // 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;
  }
}
```

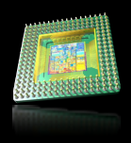


Tutorial: Adding Methods

JavaScript

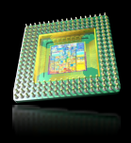
```
  // into an array that gets passed to the calculator module.
  postfixExpr = google.expr.parseExpression(
      document.getElementById('formula').value);
  var calculator_module = calculator.application.module();
  try {
    calculator_module.calculate(postfixExpr, onCalculate);
  } catch(err)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



NaCl

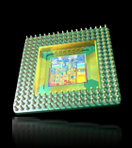
Other Demos



NaCl

Coming Attractions

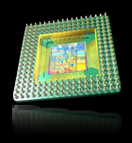
- Full debugger support
  - GDB
  - Visual Studio
- IDE plugins
  - Eclipse
  - Visual Studio
  - Xcode
- Easy to use interop library
- PNaCl (portable NaCl)



NaCl

Summary

- Choice of language!
  - C/C++, ASM, Objective-C, Python, ...
- Full power of the client's CPU
  - ...while maintaining browser neutrality
  - ...allowing OS portability
  - ...providing safety that people expect from web applications



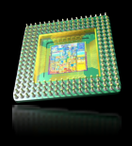
NaCl

Resources

http://code.google.com/p/nativeclient/

http://code.google.com/p/nativeclient-sdk/

http://sites.google.com/a/chromium.org/dev/nativeclient/



NaCl