

Hi. My name is Jeff Vroom and I'd like to introduce StrataCode, a code-processing tool for building complex software efficiently.



## Introducing StrataCode

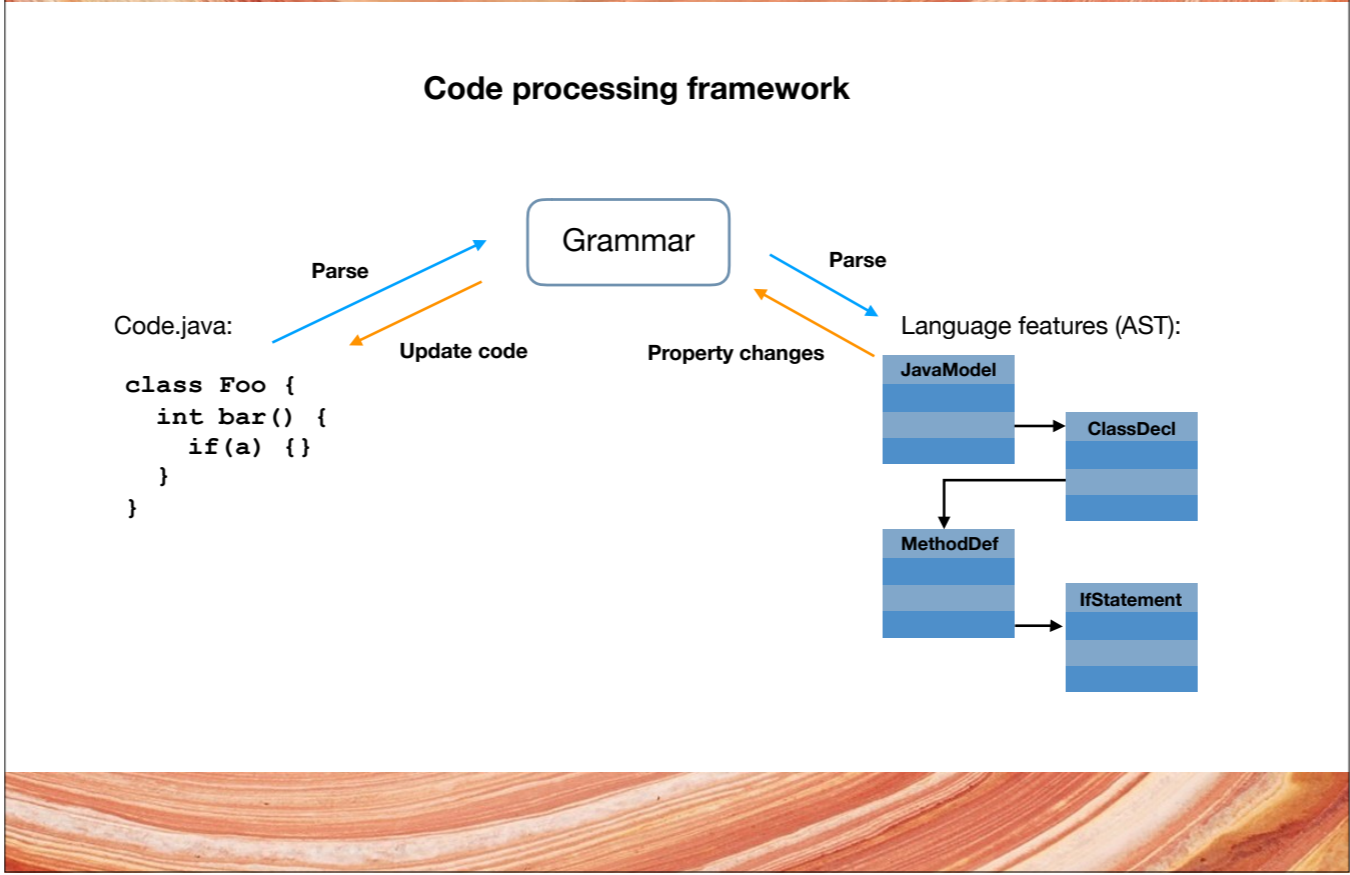
- Mostly solo project by Jeffrey Vroom
- Layers to organize code. Build declarative, efficient apps
- Software Architect: AVS, ATG, Adobe Flex Data Services
- Last ten years consulting and building StrataCode
- Looking for ideas to make it better, projects to build, and more partners to help
- Open source if there's enough support

After a long career building frameworks and applications, I started from scratch building a platform to build more declarative applications using layers to organize code.

It's been a lot of work, but I finally feel more productive building with it than anything else and am happy to share this preview. I'm looking for developers to try it out, feedback on improvements, projects to build, and more partners to help me build something awesome.

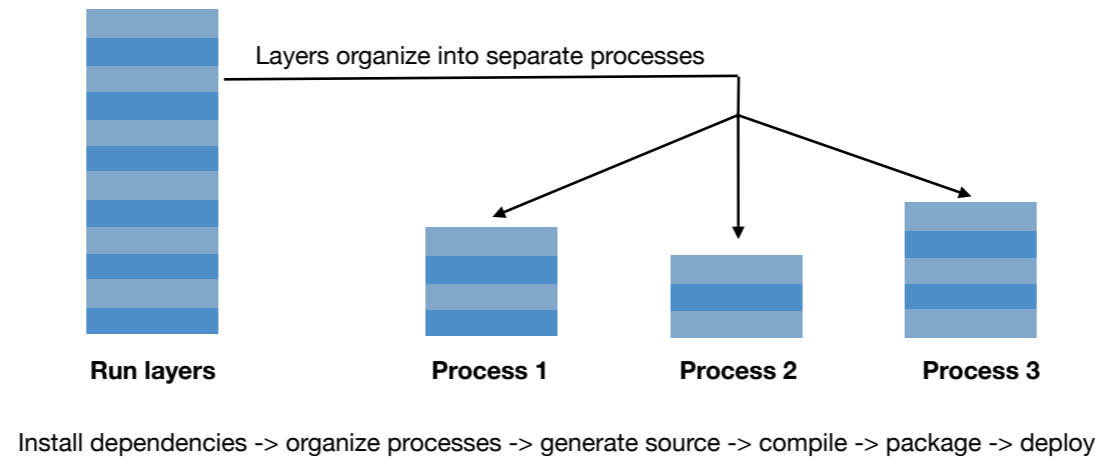
If there's enough support, it would make a great open source project.

Here's an overview of the design.



StrataCode at the lowest level is a code-processing framework that provides an easy way to add programming language grammars that read, modify, and write source. Each grammar defines an API that exposes the features of that language. When features are changed through the API, incremental updates are applied back to the source. This design makes it possible to write code that processes code a lot faster.

### Multi-process build/run from one layer stack



The code-processing framework is used in a layered build system that supports the entire lifecycle of code, from configuring, to building, testing, packaging and deploying a multi-process system.

## Live programming for better management UIs

### Code.java:

```
object temperature extends Converter {  
  value1 = 0;  
  value2 := value1 * 9.0/5.0 + 32;  
  unit1 = "Celsius";  
  unit2 = "Fahrenheit";  
  title = "Temperature";  
}
```

Refresh



Running process

Live editing

object temperature extends Converter

value1 = 0

value2 := value1 \* 9.0 / 5.0 + 32

unit1 = "Celsius"

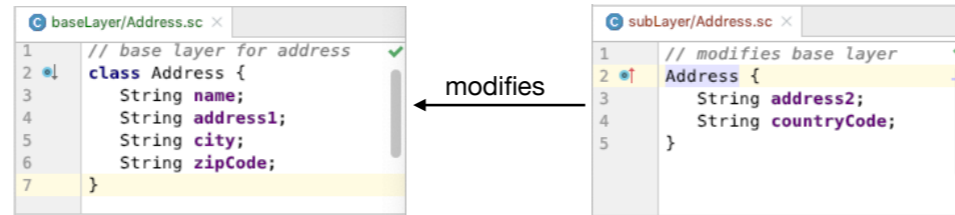
unit2 = "Fahrenheit"

title = "Temperature"

And, once an application is running, it supports refresh to pickup code changes. There's also a management UI framework that can do live-edits.

## Layered languages - extensions to Java

.sc -> StrataCode language

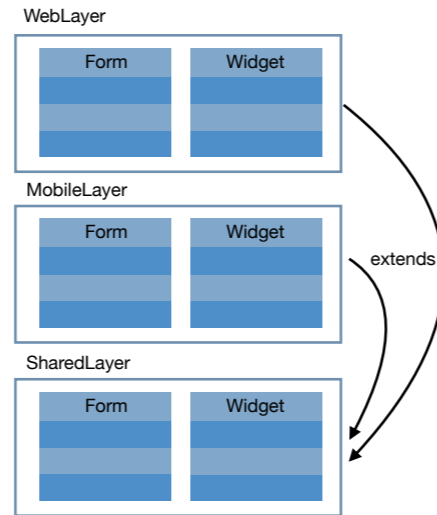


also: .sct, .shtml, .scxml, sccss, scsh formats - JSP-like integrated into the language for when a file is 'mostly text'

To get the maximum benefit of the layered architecture, StrataCode adds languages that work like Java and JSP but also support layering of types for organizing code.

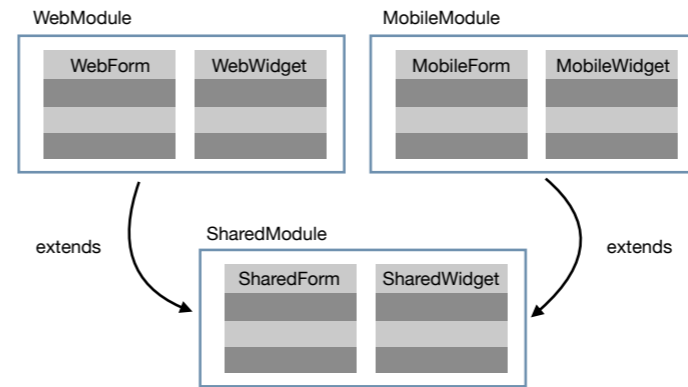
## Separating code by dependencies

### Layers using modify



- Code naturally organized by dependencies
- Better reuse, readability, refactorability

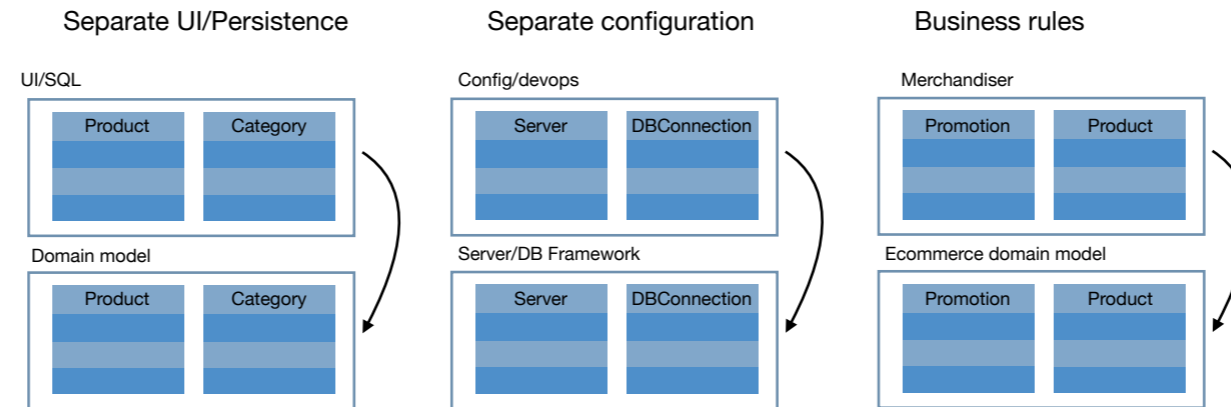
### Modules using o/o inheritance



- More, longer type names
- Wrong type in code e.g. need WebForm but have SharedForm

Layers are particularly useful for separating code by dependencies to improve reuse, allow customization, and simplify designs. Using modules, designs become more complex and are harder to refactor.

### Many uses for layers



**And more:** client/server, devops configuration + code, project configuration rules, testing, localization, style/design, plugins, inversion of control, 3rd party customizations, microservices, security sandboxes, dynamic/compiled code, A/B testing

The main reason I spent so much time building StrataCode is the potential I see for layers to improve so many aspects of software development, like separating application code from UI and persistence. And for improving the potential for customizing applications by making it easy to separate configuration and business rules as needed.



## Layered project organization

### Layer definition file

```
modelimpl.sc x html/_core.sc x schtml.sc x lib.sc x testScript.scr x IdentifierExpression.java x RemoteMethod.sc x
1 package sc.jetty;
2
3 *|* jetty.lib extends servlet.lib, log4j.core {
4   compiledOnly = true;
5   hidden = true;
6   codeType = sc.layer.CodeType.Framework;
7
8   object jettyPkg extends MvnRepositoryPackage {
9     url = "mvn://org.eclipse.jetty/jetty-webapp/9.4.7.v20170914";
10  }
11
12   object jettySchemas extends MvnRepositoryPackage {
13     url = "mvn://org.eclipse.jetty.toolchain/jetty-schemas/3.1.RC0";
14  }
15
16 *|* log4jPkg {
17   url = "mvn://org.slf4j/slf4j-log4j12/1.7.25";
18 }
19
20 *|* public void init() {
21   // Exclude the javascript, android, and gwt runtimes, inherited by default for downstream layer
22   excludeRuntimes("js", "android", "gwt");
23
24   // Jetty for now is tied to the java_Server process.
25   // TODO: move this downstream so jetty.lib is usable in other processes
26   addProcess(sc.layer.ProcessDefinition.create("Server", "java", true));
27 }
28 }
29
```

- Written in dynamic StrataCode
- Static typed, IDE support
- Improve customization intent
- Simpler project directories

And layers are an amazing way to organize projects. They offer flexible merging of directory trees, reducing the amount of copying at the start, and can override anything in a manageable way.

The layer definition file replaces typical build config and is written in dynamic StrataCode. The layer itself defines a type-safe sandbox for the code inside with default imports and annotations. Rather than a lot of project boilerplate, a layer directory may include just a few well-named files so that more team members can manage their slice of the system.

## Features for declarative programming

### Data binding

**a := b**



**a ::= b**



**a =: b**

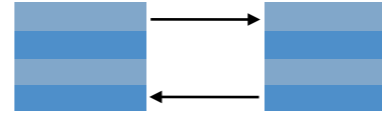


**a =: b()**



eval

### Components



init, start, validate

### Templates

- dynamic text - sct, scxml, sccss,
- build languages on top (like schtml),
- stateful and stateless support
- JSP operators, but more like an extension to Java

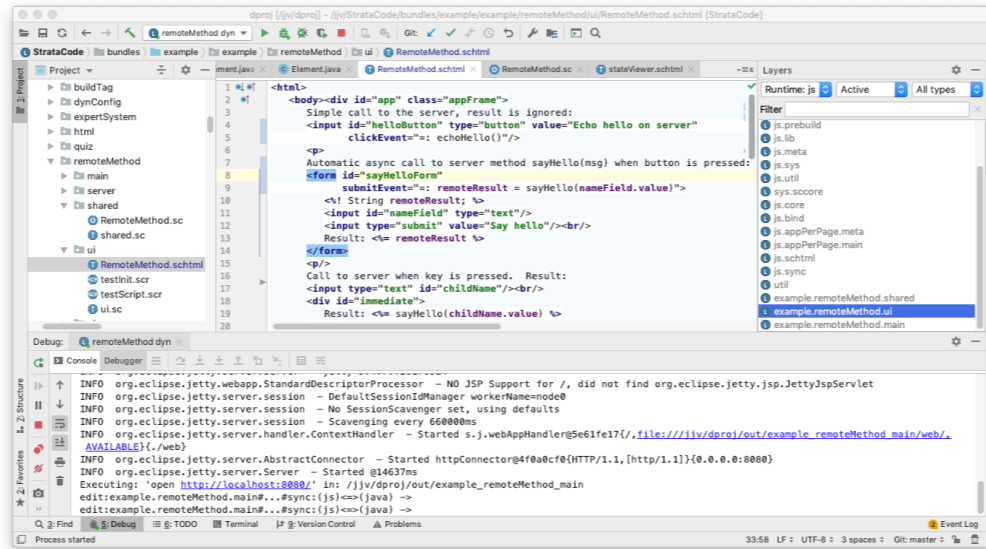
### Properties

- get/set conversion
- change events
- mix compiled and dynamic properties in one type

And for building more declarative apps, the StrataCode languages also support data binding, components, properties, and templates on top of Java. Files are converted to readable Java before compiling making it easy to debug and use in existing projects.

## IntelliJ plugin

Java-like editing, debugging for sc, sct, schtml, scj, scr in all frameworks



There's a full featured IntelliJ plugin that supports all of the StrataCode file formats and a straightforward way to add new ones.

## Code processing of language features

### **Annotations**

Perform code processing on a type when an annotation is set - 'annotation layers' for compiled Java

### **parent/child relationships**

Implement nested objects with a 3rd party library - code templates for compiled, IDynManager for dynamic

### **Full featured API, code processing engine, with runtime support**

Supports compiled or source type systems. Full type indexing for both IDE or runtime. Optional 'liveDynamicTypes' mode to track object instances of certain types for management UIs.

### **IntelliJ plugin support built in**

Usually no extra work to support framework features in the IDE

### **Much more** - carefully designed hooks for framework developers

For framework developers, there are just the right hooks using annotations, code-processors and more.

## Current frameworks

### 3rd party integrations

android, swing, junit, jdbc, servlets, opengl, opencv, jetty, jpa,

Experimental: wicket, gwt

### StrataCode web framework

There are a number of frameworks and integrations that build on top of well-known Java libraries that demonstrate these features.

## StrataCode web framework

### Rule oriented templates

```
<form submitEvent=":: addTodoEntry()">  
  <input type="text" value=":: todoText"/>  
  <input type="submit" value="Add" disabled=":: todoText.length() == 0"/>  
</form>
```

### Stateless

```
<ul>  
  <% for (int i = 0; i < 3; i++) { %>  
    <li><%= i %></li>  
  <% } %>  
</ul>
```

Converted to an output method

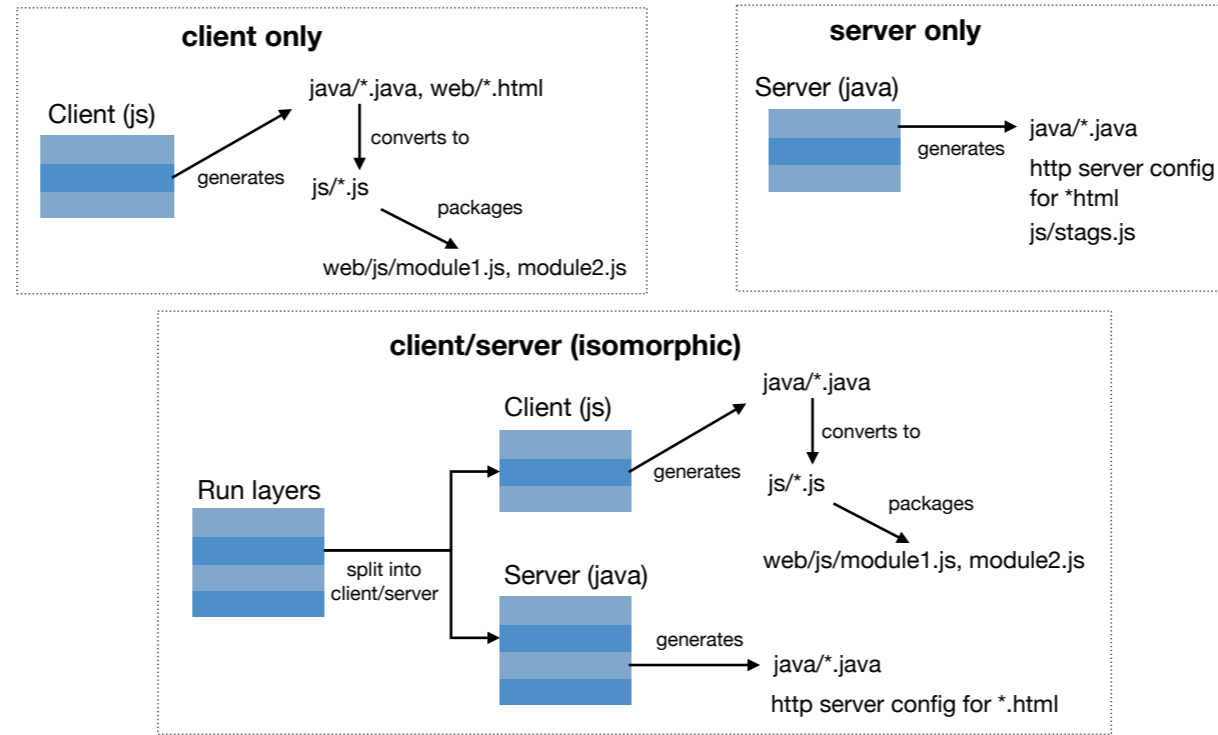
### Stateful

```
<%! ArrayList<Integer> vals = {0, 1, 2}; %>  
<ul>  
  <li repeat="vals"><%= repeatVar %></li>  
</ul>
```

Converted to reactive components

For web applications, the StrataCode web framework changes the game for Java in the browser. It supports declarative, rule-oriented web page templates, built from tag objects. Each tag object is converted into either a stateless output method, or a reactive component tree that refreshes incrementally using events.

### Three ways to deploy web components



Tag objects can run in three different ways. In client only mode, they generate static html and javascript files at build time.

In client/server mode, the server renders HTML for fast initial page loads, then javascript versions of tag objects are loaded for interactivity.

In server only mode, the server renders HTML and sends only a small Javascript file to send events to the server and receive updates to the HTML.

### Special tag attributes

**extends** - inherit attributes + body from another tag

**abstract** - define tag macros

**visible** - add/remove tag from page

**class, style** - set to expressions for dynamic logic

**repeat** - iterate tag

**replaceWith** - substitute a different tag

**DOM events** - click, mouseDown/Up/Move, keyDown/Up, focus/blur

**DOM properties** - clientWidth/Height, offsetTop/Left/Width/Height

**Merging** - tagMerge, bodyMerge, addBefore, addAfter, orderValue

**scope** - change lifecycle of the tag/page: e.g. request, window, appSession, ...

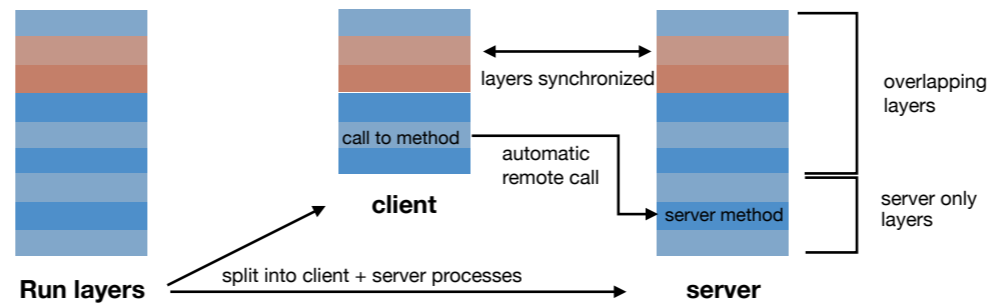
**exec** - run tag on client, server, or both - by default inherits from parent tag or app default

(and much more)

There are a number of features available as attributes for tag objects in each of the different modes.



### Data synchronization + auto RPC



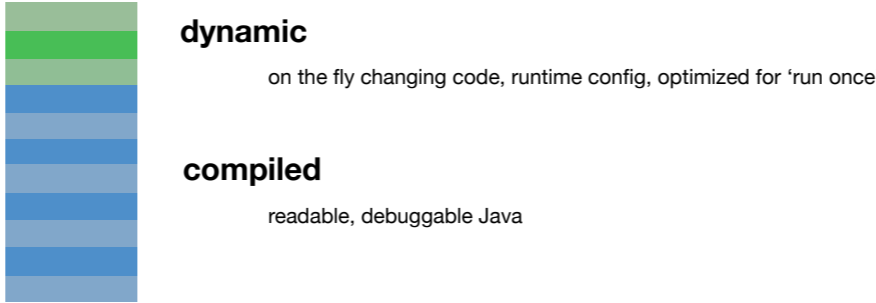
### Async call with reverse data binding

```
Automatic async call to server method sayHello(msg) when button is pressed:
<form id="sayHelloForm" submitEvent="=: remoteResult = sayHello(nameField.value)">
  <%=! String remoteResult; %>
  <input id="nameField" type="text"/>
  <input type="submit" value="Say hello"/><br/>
  Result: <%= remoteResult %>
</form>
```

Web apps can use data sync and data binding with RPC to make client/server development more seamless, and deployment style more flexible. Shared layers between client and server replace the need for a separate protocol definition. Remote method calls are detected, validated, and handled automatically using data binding, code-gen, and apis. This way of organizing code for client/server apps reduces code overall and makes it easier to change process boundaries.

### Flexible runtime that evolves and scales efficiently

One syntax - two ways to run layers and types:

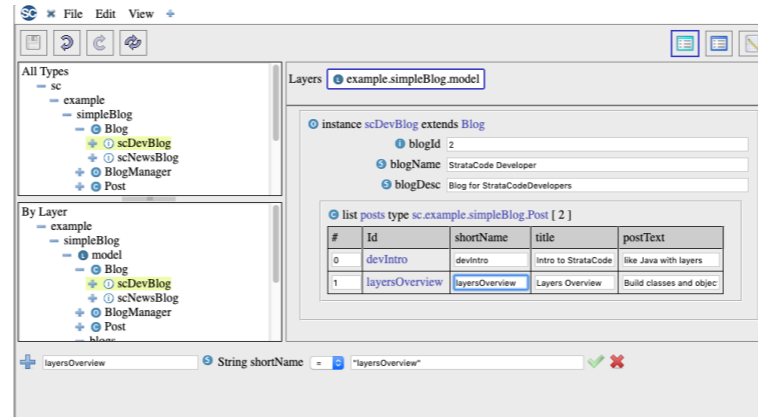


Mix compiled and dynamic features in one type - change the boundary as needed

All frameworks support dynamic types and layers for fast round trip times, even for large projects and components that have state. This feature opens the door for a whole new class of live edit management UIs and customization tools that work on large code bases.

## Management UI framework

- Build UIs from domain objects
- Portable: desktop, web
- Edit configuration, rules - in place or in a new layer
- Create instances, types, properties, layers
- Navigate by type name, by layer or both
- Layers - multiple views on the same type

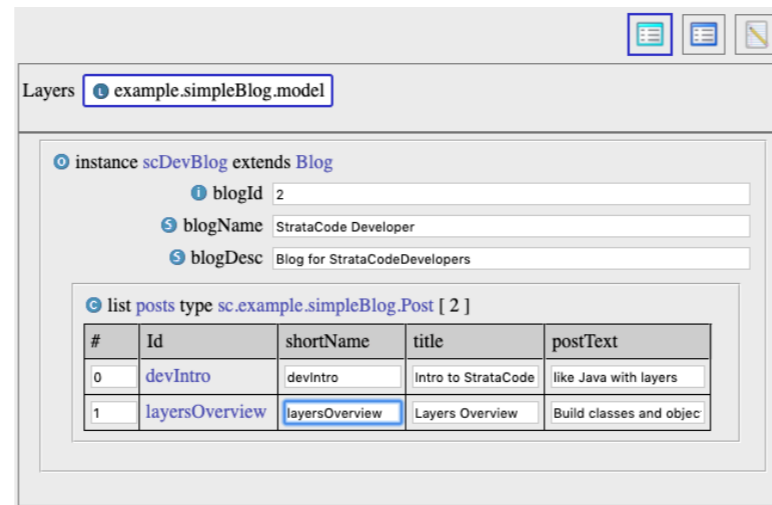


And StrataCode includes such a management UI framework that can be used with any web or swing application. It builds forms right from the application domain model types.

There are three views....

## Instance View

View, edit, create instances in the running application



Layers **example.simpleBlog.model**

**instance scDevBlog extends Blog**

- blogId** 2
- blogName** StrataCode Developer
- blogDesc** Blog for StrataCodeDevelopers

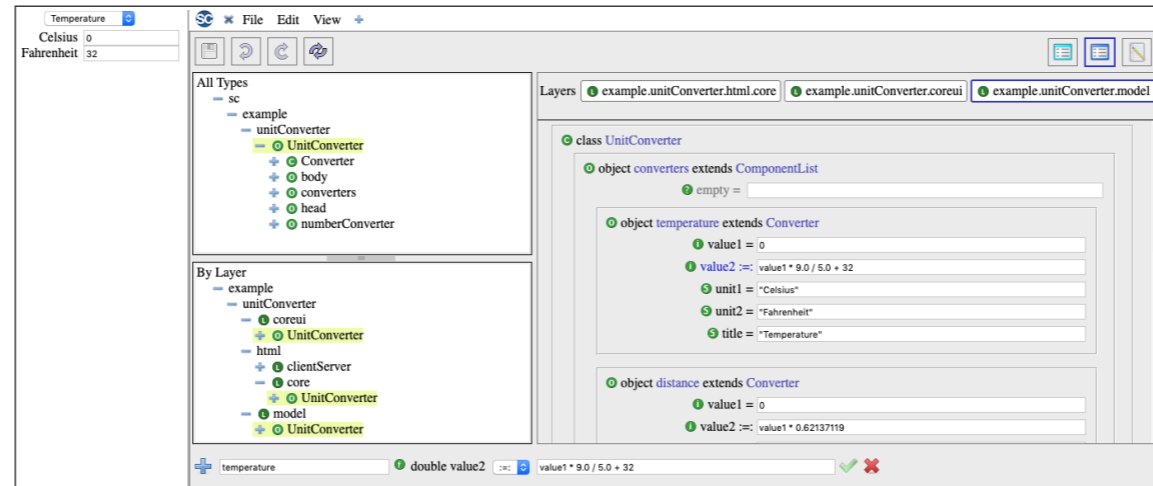
**list posts type sc.example.simpleBlog.Post [ 2 ]**

#	Id	shortName	title	postText
0	devIntro	devintro	Intro to StrataCode	like Java with layers
1	layersOverview	layersOverview	Layers Overview	Build classes and objec

Instance view allows the creation of new instances and editing of existing ones.

## Type view

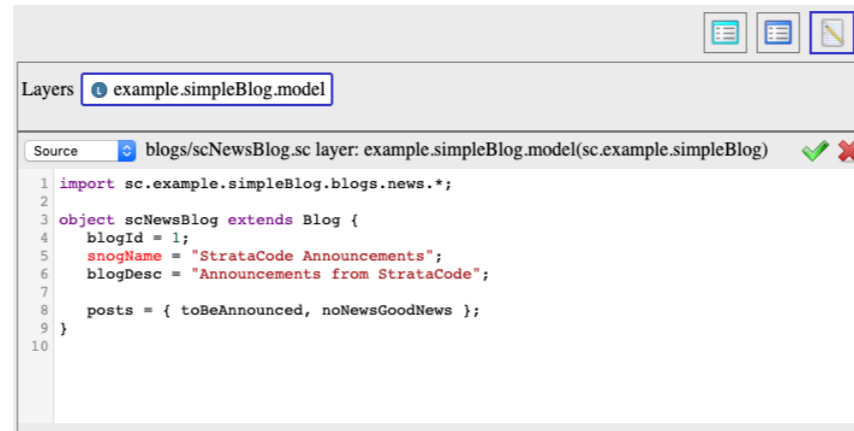
Edit property initialization, data binding rules, add properties  
 Updates source files incrementally for mixed tool/developer workflows



Type view changes configuration, edits data binding rules, and adds new properties on the fly. Change code in-place or adding the change to a new layer.

## Code view

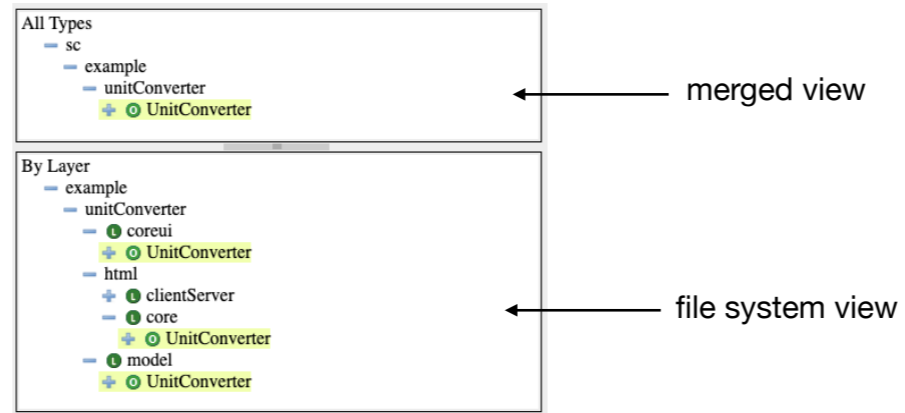
Mini IDE (using codemirror in the browser, rtext in swing)  
Edit-time errors, syntax highlighting, code-hinting



```
1 import sc.example.simpleBlog.blogs.news.*;
2
3 object scNewsBlog extends Blog {
4   blogId = 1;
5   snogName = "StrataCode Announcements";
6   blogDesc = "Announcements from StrataCode";
7
8   posts = { toBeAnnounced, noNewsGoodNews };
9 }
10
```

Code view edits source code directly using a mini-IDE.

### Navigate by type or by layer



There are two ways to navigate types and instances in the management UI: by type name, or with layers, showing the file organization.

## Versatile test scripts, command line

```
Product.sc x TypeEditor.java x testEditBooks.scr x testToDoList.scr x model/ToDoList.sc x - 81s
1 cmd.pauseTime = 250;
2
3 // Run these commands in the java process - they will be sync'd to the client.
4 cmd.targetRuntime = "java";
5 // Optionally - send the commands to the client - after converting to JS,
6 // they are eval'd using the sync framework.
7 //cmd.targetRuntime = "js";
8
9 ToDoList {
10   todos.size();
11   assert todos.size() == 3 : "No todos";
12   todos.get(1).complete = true;
13
14   todoText = "A new entry";
15   addTodoEntry();
16   removeComplete();
17
18   while (todos.size() > 0) {
19     todos.get(0).complete = true;
20     removeComplete();
21   }
22
23   todoText = "done";
24   addTodoEntry();
25   cmd.sleep(1000);
26   todos.get(0).complete = true;
27 }
28
```

- Line-oriented StrataCode
- IDE support
- Target one or more processes
- Automatic remote methods
- Layering, nesting with 'include'
- Script mode - edit instances
- Edit mode - edit types

Use the scr format for test scripts or for command-line control over a multi-process app.





## Learn more

Learn more at [www.stratacode.com](http://www.stratacode.com)

Contact [jeff@jvroom.com](mailto:jeff@jvroom.com)

See the status page for how we are doing

Examples, documentation, articles

Ideas for improvements?

Build something together?

That's the end of the overview. Check the website for current status and more information.

Let me know what you think and if you or are interested in teaming up to build something awesome with StrataCode.

Thanks for watching.