SmartClient Training

Become productive & start building cutting edge applications

Isomorphic SOFTWARE

Training Course: Introduction to SmartClient

The Introduction to SmartClient training course is designed to get you up and running, productive, and able to start building powerful, cutting-edge applications. It typically lasts three full days, spanning a weekend (Thurs, Fri, Mon) OR six half days. This allows time for you to try out what you have learned, and be able to ask more informed questions on the final day.

Course topics are:

- ✓ Java / JavaScript and CSS Prerequisites
- ✓ Installation & Deployment
- ✓ SmartClient Class System
- ✓ UI Components
- ✓ Data Binding
- ✓ Developer Tools
- ✓ Data Integration
- ✓ Metadata Management
- ✓ Optional Modules
- ✓ Client-Server Communication
- ✓ Event Handling
- ✓ Basic Branding
- ✓ Extending UI Components
- ✓ Performance
- ✓ Animation
- ✓ JPA, Hibernate, SQL Connectors
- ✓ Custom Server Data Integration

> The training was excellent ... we were in awe of Paul's technical ability, he just seems to know everything."
>
> *Northrup Grumman*

Register Now

http://smartclient.com/services/index.jsp#training
Courses do fill up. Early booking will secure your place.

DETAILED AGENDA

#1 Welcome

a) Brief overview of Isomorphic Software

b) What is SmartClient

#2 Introducing the SmartClient SDK

a) Installation

SDK directory structure includes smartclientRuntime and smartclientSDK directories.

- smartclientRuntime is for deployment – includes required runtime resources only
- smartclientSDK is for development and prototyping - includes
    - embedded servlet engine (apache tomcat)
    - embedded SQL database (HSQLDB)
    - extensive Documentation and Examples

b) Exploring the SDK

- Home Page
- Documentation (QuickStart guide, SmartClient Reference, Module-specific PDFs)
- Feature Explorer
- Tools including Admin Console and Visual Builder

c) Deploying and using SmartClient

- Deployment (See also "Deploying SmartClient" topic in SmartClient Reference)
- Loading SmartClient
    - "templates" directory in SDK
    - how make a SmartClient-enabled page

#3 Coding in SmartClient

a) Hello World example – how to create a SmartClient component.

- isc object
- class.create()
- methods and properties
- global ID
- Colliding ID's / introduction to Developer Consult

b) Introduction to SmartClient classes

- Canvas class
- Inheritance in SmartClient
- The Super method
- init and initWidget
- Inheritance and documentation

c) Declaring components in XML

- Requires SmartClient server
- Functionally equivalent to JavaScript – and ultimately translated to JS
- <JS> tags and <![CDATA[blocks

DETAILED AGENDA

#3 Coding in SmartClient (Continued)

d) Creating SmartClient subclasses

Uses of custom class vs instance properties

class.addProperties() and class.addClassProperties()

* Revisited later

Exercise 1

- copy isomorphicSDK/templates/helloWorld.jsp to SmartClient/scratch.jsp
- access as http://localhost:8080/scratch.jsp
- open in text editor of choice
- modify scratch.jsp: build class "SayButton" that say()s this.message when clicked
- use this file for all further exercises

#4 SmartClient User Interface Components

a) Canvas

- Base UI Component
- Simple properties:
  - contents
  - backgroundColor, border, styleName
  - sizing, positioning, overflow
- draw() / show()
- Events
  - click, doubleClick, right click, dragResposition
  - focus and key events

b) Grids

- ListGrid
  - Fields and Records (data List - array)
  - Events
  - User Interactions
    - sorting
    - editing
    - filtering (query by example)
    - grouping
    - freezing fields
    - header drag
    - auto fit
    - selection
    - incremental data loading
  - Additional features / customization
    - field type display (text, images, links, dates)
    - custom cell values, styling
    - printing
    - nested components
    - formula / summary fields fields
    - grid / group summaries
    - advanced filter
    - View state
  - Header and body properties
  - Introductions to AutoChild concept

DETAILED AGENDA

Exercise 2

- create a grid with two fields "TextValue", "Remove" and two rows of data
- clicking in "Remove" field should delete the record
- Remove field should show an “X” character in it (should not be included in record data)

#4 SmartClient User Interface Components (Continued)

- TreeGrid
  - Inherits from ListGrid
  - Hierarchical data (Tree object)
- CubeGrid
  - (Part of the optional Analytics module)
  - Data Structure:
    - No Fields: Every cell is a record
    - Facets and FacetValues
    - Record objects and data loading

c) Forms

- DynamicForm vs HTML Form
- Fields and values (single records)
- Item types
  - data items vs control items
  - data type and editor type
  - valueMap and valueIcons
  - canvasItems
- Appearance
  - layout, hint, titles, icons, valueIcons
  - inline icons
  - cellBorder for debugging
- Features
  - appearance (layout, titles, hint, icons, hover)
  - show / hide, enable / disable
  - validation
  - events
  - mask / keypressFilters
- ValuesManager
  - Similar APIs to forms but allows separated presentation

d) Additional Data Components

- TileGrid
  - SimpleTile
  - TileRecord
- ColumnTree (Miller Columns)
- DetailViewer
- Calendar
- Charts (Optional module)
  - FacetChart vs FusionChart
  - explicit creation vs DataBoundComponent chartData() API
  - chartType
  - data model
    - facet-based, similar to cubeGrid
    - inlined facets (values declared within each facet)
    - support for multiple facets, to display related values on a chart
    - may be bound to a DataSource
  - Zoom
  - User interactions: pointClick, valueClick, showValueOnHover, hoverHTML
  - DrawPane subclass - can use drawing APIs to customize appearance
  - Charts may be exported in PDF or image format

DETAILED AGENDA

#4 SmartClient User Interface Components (Continued)

e) Layout Components

- Nesting of components.
- Canvas children
  - positioning, sizing, overflow
- Layout members
  - Stack vs Layout
  - Nesting Layouts
  - LayoutSpacers
  - margins, alignment
  - member resizeBars
  - dragReposition within layouts
    (acts as a drag/drop, required canAcceptDrop)
- Windows
  - items
  - header / footer controls
  - modality
  - events (minimize, close)
- Tabsets / Tabs
  - tabs and panes
  - align / orientation
  - selection events
  - tab bar controls
- SectionStacks
  - sections and items
  - visibility mode
  - interactions (expand, drag)
  - section customization (showHeader, title, icon, header controls)
- TileLayout / FlowLayout
  - Tiles
  - Superclass of tileGrid
- SplitPane
  - designed for navigation / list / detail data presentation
  - navigationPane, detailPane, listPane
  - device-dependent appearance (revisited in “Mobile Development”)
  - navigatePane / autoNavigate
  - showDetailPane() / showListPane() / showNavigationPane()
  - dynamic pane titles (see listPaneTitleTemplate)

f) Control Components

- Buttons
  - Statefulness and styling
  - ImgButtons, StretchImgButton, IButton class
- Menus
  - menubutton / menubar vs context menus
  - dynamic content (Title, checkmark, enabled/disabled)
  - custom fields (inherits from ListGrid)
  - shortcut keys
  - submenus
  - tree menus/ tree menu item
- Other control components: Slider, ToolStrip

DETAILED AGENDA

Exercise 3

- Create a Button
  - that creates a Window with title "My Window"
- containing a TabSet with a single Tab "My Tab"
  - containing a DynamicForm
    - with a button and text input in a row.
- Clicking on the button should append "Foo" to the current value of the text field.

Suggestion: start with either the DynamicForm or the Window and add each additional component one at a time.

#4 SmartClient User Interface Components (Continued)

g) Drawing

- standard API for drawing shapes beyond traditional HTML
- approaches: HTML5 <canvas>, SVG, VML
- DrawPane class
- DrawItem subclasses

Note: FacetChart makes use of the Drawing module

#5. SmartClient System Utilities

(Note: Not an exhaustive list of all utilities)

- Extensions to native JS objects (String, Array, Date, Number, etc)
- DateUtil, NumberUtil
- JSON, XMLTools
- isA
- History
- Offline

#6 Extending SmartClient

a) SmartClient components and inheritance

Existing components

- superclasses
- automatically created children

b) Creating custom components

See “Component Reuse” example in Feature Explorer

- Creating children at runtime
  - initWidget override
  - use property names rather than global IDs
  - passing properties through to children
- Embedding Custom HTML, including 3rd party widgets
- Auto Children subsystem
  - addAutoChild vs createAutoChild

c) ComponentXML

- XML format for components - used by (but not restricted to) Visual Builder
- screenLoader servlet / RPCManager.loadScreen()
- accessing loaded components in Java code
- dynamic screen generator

d) Dashboard and Tools, EditMode

- Support for user-driven UI creation and modification

DETAILED AGENDA 6/14

#7 Data Binding

a) Introduction to DataSources

- What is a DataSource
- How to define / load a dataSource
  - Inline Java definition
  - XML definition (requires SC server)
    - loadDS jsp tag
    - <script src=...> tag pointing to dataSourceLoader servlet
- Where does data come from (overview only)
  - client only dataSource
  - JSON / XML dataSource
  - SQL DataSource

NOTE: DataSource APIs are unchanged on the client for different data integration strategies. You can swap a client only dataSource with a SQL or REST dataSource with no impact on the components and code that accesses the data.

b) DataSource fields

- field properties
  - name
  - type
  - title
  - hidden
- PrimaryKey field
- foreignKey field (for hierarchical data)
- validators
  - type
  - required
  - built in validator types

c) Explicit DataSource APIs and related concepts

- Allows for direct data fetch / manipulation outside the standard databound components
- fetchData() / addData() / updateData() / removeData()
- callbacks and DSRequest / DSResponse objects
- Operation Types / Operation Bindings

d) Component binding

- Common databound components:
  - DynamicForm, ListGrid, TreeGrid, DetailViewer, TileGrid
- Combining component and dataSource fields
- Databinding APIs and behaviors:
  - ListGrid / TreeGrid:
    - fetchData() / filterData() / autoFetchData / invalidateCache
    - criteria objects
    - filterEditor
    - incremental data loading, server side sort
    - invalidateCache()
    - Create / Remove / Update / Delete [CRUD]:
      - canEdit, startEditing() / startEditingNew()
      - saving and autoSaveEdits
      - pending edit values
      - validation
    - removeData / removeSelectedData
  - DynamicForm:
    - editRecord() / editNewRecord()
    - saveData() / saveOperationType
    - getting values as criteria
    - AdvancedCriteria and Filter Builder
  - FormItem valueMaps / optionDataSource
    - valueField, displayField, picklistFields
  - DynamicForm and ValuesManager

DETAILED AGENDA

#7 Data Binding (Continued)

e) Data Model objects

- ResultSet / ResultTree
  - automatically generated
  - List Interface
  - Intelligent cache management

Exercise 4

- Create a DataSource in JavaScript, with a DynamicForm and a ListGrid bound to it. Using only settings on the DataSource:
  - make the form show one Text Field titled "Name" and one drop-down select titled "Occupation" with values "CEO", "CTO" and "CFO".
  - The grid should also show two columns, "Name" and "Occupation"

#8 Data Integration

See documentation topic "Client-Server Integration", and sub topics

a) Client-side databinding

- Client-only dataSource - client side test data
  - Note asynchronous operations, standard DS cache mgmt
- Fetching / Updating remote data by URL
  - XML / JSON operation
  - dataURL, recordXPath, valueXPath
  - dataFormat / dataProtocol
  - operationBindings for per operation URL, protocol etc.
    - (operationType / operationID)
  - transformRequrest / transformResponse
  - cacheAllData / testFileName / dataURL for clientOnlyDS
  - RestDataSource class
    - per operation dataURLs
    - support for meta data (start row, end row etc)
    - Documented format for server inbound data and responses
  - Existing web services -- SchemaSet loadWSDL / loadXMLSchema (see WsdlBinding topic)
- Options for totally custom client-side data integration:
  - ClientOnlyDataSource + getClientOnlyResponse
  - "clientCustom" operationBinding dataProtocol
  - + transformRequest / processResponse

Exercise 5

- copy examples/shared/ds/test_data/animals.data.xml into a new subdirectory of smartclientSDK/
- construct a DataSource that can read this data in response to a fetch
  - show a grid bound to this dataSource fetching data
  - show a form bound to the same data source such that selecting a record in the list grid shows the values in the form

DETAILED AGENDA

#8 Data Integration

b) Server-side databinding

- Generic server dataSource / server side features:
  - Define dataSource on server using ds.xml file
  - IDACall servlet
  - generic ds approaches (for arbitrary business logic)
    - BasicDataSource subclass plus serverConstructor attribute.
      - Implement execute... methods
      - Server-Side DSRequest / DSResponse objects

Example:
/isomorphic/system/reference/SmartClient_Explorer.html#customDataSource
Note - extensible format (example "mapped bean class"):

/isomorphic/system/reference/SmartClient_Explorer.html#ormDataSource

DMI dataSource
- dataSource.serverObject, operationBinding.serverObject, serverMethod
- method signature / return type
example in SDK: DMI example under: /examples/server_integration/

Server side scripting
- dataSource.script / operationBinding.script
- languages
- context variables (see "Velocity variables") and return values

Example:
/isomorphic/system/reference/SmartClient_Explorer.html#scriptingUserSpecificData

Notes:
- operationType "custom" for arbitrary (non crud) operation
- Server code can instantiate DSRequests
- Server code can also (eg) issue http requests against another server, so could use a server-side ds to integrate with a web service

- Server side validation
  - type / built-in validation runs on client and server
  - custom validation by setting properties on DSResponse

- Velocity support
  - allows you to directly customize dataSource behavior by injecting VTL statements to be evaluated on the server at runtime.

- OperationBinding features:
  - DMI serverObject, methodName, method arguments
  - declarative authorization/authentication (requiresAuth / requiresRole / requires)
    - JAAS integration via httpServletRequest.getRemoteUser() and isUserInRole()
    - Non-JAAS support via servlet override / rpcManager.setUserRoles()
  - outputs
  - mail

- autoDeriveSchema + schemaBean

- Queued requests and "transaction chaining"

DETAILED AGENDA

#8 Data Integration (Continued)

Examples under 'transactions' in feature explorer

- SQL DataSource (See SQLDataSource topic in docs package)
  - serverType= "sql"
  - database config - server.properties + ds tableName
  - default SQL behavior
  - custom SQL with no code
    - criteria / values objects
    - custom sql templating:
      - customSQL
      - selectClause, whereClause, valuesClause, tableClause
      - field properties:
        - customSQL
        - tableName
        - sqlStorageStrategy
  - Combining with DMI to allow custom java code
    - totally custom operations
    - running custom behavior before / after default logic
    - injecting custom velocity variables
  - autoDeriveSchema + tableName
  - Database joins: includeFrom / includeVia
  - DataSource.audit (Not strictly limited to SQL DataSources)

- Hibernate DataSource
  - HibernateIntegration doc topic
  - JPAIntegration doc topic
    (See "Persistence Technologies" documentation group)

- REST DataSource Servlet
  - Allows access to any server-side dataSource via the documented client-side RESTDataSource request / response formats.
  - This means you can access SmartGWT dataSources on the server from any client side technology that via HTTP.

- DynamicDSGenerator
  - Allows DataSource creation on the server at runtime

Exercise 6

- Modify the supplyItem.ds.xml file to have an additional fetch operation which only returns records where the unitCost is less than 1.
  - Hint: This could be achieved via customized SQL, or by modifying the inbound request
- Create a ListGrid bound to this new dataSource, showing the data returned by this operation

c) DataSource "offline" support

DETAILED AGENDA

#9 Client-Server Communication

a) Introduction to the RPCManager class

- RPCManager handles low level client/server communications
- Used by DataSource code
- Can be accessed directly via documented APIs

b) Common use cases for direct RPCManager APIs

- Start/send queue
- default actionURL (servlet)
- send / sendRequest non "record" data
    - o atomic responses [yes/no]
    - o unstructured data (HTML content, code)

c) RPCManager Features

- automatic, bi-directional, type-safe Java<->JavaScript translation of nested structures
- http proxying
- request queuing

d) Additional client-server communication functionality

- DMI class + app.xml configuration file

Example:
/examples/server_integration/index.html#genericRPCIntegrationDMI

- WebService class

e) Additional client-server communication functionality

- DMI class + app.xml configuration file

Example:
/examples/server_integration/index.html#genericRPCIntegrationDMI

- WebService class

f) Relogin

- goal: seamless handling of user's privileges expiring, integrated with standard server authentication
    - o achieved via special server responses
    - o options for handling (RPCManager.loginRequired(), resubmit transaction)

DETAILED AGENDA

#10 Developer Tools

a) Developer Console

- Launching the developer console
- Overview of tabs
- The DOM inspector
- Running code from the Eval area
- Logging Categories and priorities
- Adding logging messages to applications
- Errors and Stack Traces
- Remote Debugging
- "Debug" non-minified / obfuscated modules
- Error reporting / SmartClient forums

* Debugging topic in SmartClient Reference

b) Server logs

- Where to view server logs
- How to customize logging sensitivity

c) The Visual Builder

- Building application views
- DataSource wizard
- Loading and Saving code

d) Admin Console

- Database configuration
- DataSource import

e) DataSource Generator

f) Automated testing Support

- Selenium integration (See "Automated Testing" and "Using Selenium" doc topics)
  - Selenium RC vs Selenium WebDriver
  - AutoTest class / SmartClient Locators
  - Handling asynchronous application state changes (waitForElementClickable etc)
- TestRunner class to manage automated playback of recorded suites

Exercise 7

- Use Visual Builder to create a grid, form and save button, such that:
  - the grid and form are side by side
  - the button is beneath the form
  - the grid and form are bound to the supplyItem DataSource
  - clicking on the grid populates the form
  - clicking the button saves the form

DETAILED AGENDA

#11 Event Handling

a) SmartClient event model

- SmartClient event types
- Retrieving event details via EventHandler class
- SmartClient event bubbling
- Canceling events

b) Page level events

- Capturing events at the Page level
    - o setEvent() / clearEvent() / registerKey() / unregisterKey()
    - o cancelling events
- Page-only events – load, unload, resize, idle

c) Event Masking

- modal SmartClient components
- canvas.showClickMask() API
    - o "soft" vs "hard" masks
- canvas.showComponentMask() API

#12 Mobile Development

(See "mobile development" documentation topic in the Reference docs)

a) Adaptive or "mobile-focused" components

- Form controls (SelectItem, ComboBox, Menus)
- SplitPane
- Navigation bar

b) Touch Event handling

c) Detecting mobile views

- Browser.isTouch, Brpwser.isHandset, Browser.isTablet
- Page.getOrientation()

d) Remote debugging

e) Packaging SmartClient applications as native apps.

DETAILED AGENDA

#13 Customizing Appearance, Look and Feel (skinning / branding)

a) Localization (See i18n documentation topic)

- Framework localization language packs
- Application message localization
- Datasource localization

b) Display density

- Canvas.resizeControls() / Canvas.resizeFonts()

c) Skins available in the SDK

- See isomorphic/skins directory

d) Loading skins

- Loading via script tag vs skin attribute on loadISC tag

e) Anatomy of a skin directory

- load_skin.js
- skin_styles.css
- images directory

f) Creating a custom skin

- start with existing skin
- component images and styles
- loading your custom skin
- remove / recreate compressed versions

See “skinning” topic

#14 Performance

a) Simple optimizations

- Understanding create() vs draw() vs show()
- Ensure global autoDraw is disabled
- Consider lazy creation of initially hidden user interfaces
- Component reuse (see SmartClient Architecture doc topic)
- Memory reclamation, understanding canvas.destroy()
- canvas.redraw() vs canvas.markForRedraw()
- RPCRequest queuing
- ListGrid optimization
    - draw ahead ratio
    - resultSet.resultSize
    - record component pooling

b) Network Usage

- SmartClient requirements
- Browser caching
    - version parameter on resource URL (automatically applied when using taglib)
    - version-specific skinImgDir
    - FileDownload servlet
- Compression
    - gzip resources and FileDownload Servlet
    - dynamic compression – CompressionFilter
- FileLoader Overview
    - What is it?
    - How to use it

DETAILED AGENDA

#14 Performance (Continued)

c) Loading application resources

- ViewLoader
- RPCManager loadScreen/cacheScreen for screens
- HTMLFlow / HTMLPane *NOT FOR LOADING SMARTCLIENT PAGES!

#15 Optional Modules

- Analytics
- Charting
- Real-Time Messaging
- Network Performance

#16 Q&A

HELPFUL RESOURCES

During and after training, you will find the following sources of information very useful:

SmartClient website: http://www.smartclient.com

- Public SDK downloads: http://www.smartclient.com/product/download.jsp
- Documentation: http://www.smartclient.com/product/documentation.jsp
- SmartClient support forums: http://forums.smartclient.com/
- Contacting SmartClient: support@isomorphic.com or http://www.smartclient.com/company/contact.jsp
- SmartClient blog: http://blog.smartclient.com/

SmartClient SDK deployed at localhost:8080

- QuickStart guide: http://localhost:8080/docs/SmartClient_Quick_Start_Guide.pdf
- SmartClient Reference: http://localhost:8080/isomorphic/system/reference/SmartClient_Reference.html
- Feature Explorer: http://localhost:8080/isomorphic/system/reference/SmartClient_Explorer.html

SmartClient Training

Isomorphic Software
1 Sansome Street, Suite 3500
San Francisco, CA 94104, USA