Smart GWT Training

Become productive & start building cutting edge applications

Isomorphic SOFTWARE

Training Course: Introduction to Smart GWT

The Introduction to Smart GWT 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
- ✓ 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?

c) What is Smart GWT?

#2 Setting up your machine for Smart GWT / SmartGWTEE development

a) Installation

Requirements:

- Java JDK
- GWT
- SmartGwtEE (useful to have SmartGwt LGPL too for samples)
- apache ant (included in SGWTEE download)

For ease of development we recommend:

- Eclipse
- GWT Plugin for Eclipse

b) Resources

Documentation:

- Sample index.html
- JavaDocs (online and local)
    - o note the special “docs” package
- per-sample readme files
- documentation embedded in live sample

Support:

- Forums (include some sticky starter-guides / how-to's)
    - o note the “FAQ’s” topic
- (support@isomorphic.com)

DETAILED AGENDA

#2 Introducing the SmartClient SDK (Continued)

c) Running the samples

- Can use ant to run directly from command line
- Import sample project(s) to Eclipse and run using the GWT plugin.
- Note: Showcase war also included for simple deployment

d) Anatomy of a Smart GWT project

- Src directory
- War directory
    - o Startic versus generated content

Note that the GWT site http://code.google.com/ has extensive docs GWT projects

- Setup / troubleshooting steps to import a sample into Eclipse:
    - o Use the “File” / “New Java Project” option to create a new project
    - o Instead of the default location, browse to the existing sample directory and import. (A number of warnings will show up. This is usual.)
    - o Use the “Properties” context menu item to edit the project:
    - o Ensure the Google / Use Google Web Toolkit option is enabled with an appropriate version of GWT.
- Can use the latest version available, supplied by the GWT plugin, or a downloaded version.
- 2.7.x is recommended for “SuperDevMode”, but not essential
    - o Under Java Build Path, set up the SGWTEE_HOME environment variable to point to the downloaded package. Used to retrieve the jars on the classpath.
    - o Use the “Run As” / “Run Configurations…” context menu to customize the GWT run configuration for the project. The “Arguments” tag allows various options when you run the app locally. I recommend increasing the available VM memory to 1G.
    - o To launch the app, use the context menu “Run As…” option to view as a web app in Classic or SuperDev Mode (more on this later).

DETAILED AGENDA

#3 Coding in Smart GWT

a) Creating a first application (See also: /docs/SgwtEESetup.html in javadoc)

- 2 possibilities:
  - o Create a new GWT application and add SmartGWT functionality to it
  - o Copy an existing sample.

We usually recommend starting with an existing sample for a SGWTEE app unless you already have a running app you are adding SGWT functionality to.

However, steps to create a new Smart GWT app from scratch:
SmartGWT (client side functionality only):

- create a gwt application
- add smartgwt libraries to build path
- inherit the SmartGwtEE module(s)

```
<inherits name="com.smartgwtee.SmartGwtEE" /> or
<inherits name="com.smartgwt.SmartGwt" />
(Includes SmartGwt / SmartGwtEE functionality and Enterprise skin).
No script version:
<inherits name="com.smartgwt.SmartGwtNoScript" />
 Specific themes:
<inherits name="com.smartclient.theme.enterprise.EnterpriseResources" />
 Tools:
<inherits name="com.smartgwt.tools.SmartGwtTools"/>
```

- modify bootstrap file. Set isomorphic dir variable

SmartGWTEE

- start by importing a sample with similar functionality to what you need.
- include all jars under SGWTEE_HOME/lib for complete server-side functionality
- merge or copy the following resources to your project

(modifying app names if appropriate)
web.xml, taglib, server.properties, log4j config, and all WEB-INF/lib jars
*See also: /docs/SgwtEESetup.html in javadoc

b) SmartGwt Classes and components:

- existing classes / APIs
  - o Widget classes (E.G: Canvas) and utility classes (E.G: SC)
- SmartClient (JS) vs SmartGwt (Java)

DETAILED AGENDA

Note: Exercises may require APIs not explicitly covered up to this point. The SmartClient Reference and the existing examples in the Feature Explorer are an excellent source of additional information.

Exercise 1

Reminder: to create a new SmartGWTPower project from built-in-ds:
(Also see the readme.txt file per sample)

- *copy the built-in-ds sample (in your file system)*
- *in Eclipse select New > Java Project*
- *select "from existing source" and browse to the copied project*
- *under project / properties*
    - *enable google/use gwt*
    - *add the SGWT_POWER_HOME environment variable (import all the .jars from the lib dir of the package)*

- Create a new SmartGwtEE project called "Scratch", using the "built-in-ds" sample as a template
- build a class "SayButton" that displays its "message" property when clicked.
- use this project for all further exercises

DETAILED AGENDA

#4 SmartClient User Interface Components

a) Canvas

- Base UI Component
- Simple properties:
    - o contents
    - o backgroundColor, border, styleName
    - o 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
        - grid / group summaries
        - view state
        - advanced filter
    - Header and body properties
    - Introduction to AutoChild concept

- TreeGrid
    - Inherits from ListGrid
    - Hierarchical data (Tree object)

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)

DETAILED AGENDA

#4 SmartClient User Interface Components (Continued)

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
- 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
- ColumnTree (Miller Columns)
- DetailViewer
- Calendar
- Charts (Optional module)
  - 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:
    - contextMenu, pointClick, valueClick, showValueOnHover, hoverHTML
  - DrawPane subclass - can use drawing APIs to customize appearance
  - Charts may be exported in PDF or image format

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
- 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)
- 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)

#4 SmartClient User Interface Components (Continued)

f) Control Components

- Buttons
  - action, click
  - 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

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.

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 Extending Smart GWT

a) SmartGWT components and inheritance

- Existing components
  - superclasses
  - automatically created children (auto-child pattern)

b) Creating custo components

- Applying customizations in constructor
- Component lifecycle
  - JavaScript widget creation
  - implicit vs explicit component creation, isCreated()
  - configuration properties (such as global ID)
  - initHandler, drawHandler
  - reclaiming memory (destroy(), markForDestroy(), isDestroyed(), explicit dereferencing)

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
- registering classes for reflection (see also DataSourceField.editorType)

d) Dashboard and Tools, EditMode

- Support for user-driven UI creation and modification

DETAILED AGENDA

#6 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

#6 Data Binding (Continued)

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

e) Data Model objects

- ResultSet

- List Interface
- Automatically generated
- Intelligent cache management

- ResultTree object

- Tree interface

Exercise 4

- Create a DataSource in directly Java, 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"

DETAILED AGENDA

#7 Data Integration

See documentation topics: “ClientServerIntegration”, “ClientDataIntegration”, “ServerDataIntegration”, “WritingCustomDS”

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

Examples:
http://www.smartclient.com/smartgwt/showcase/#grid_dataoperations_fetch
http://www.smartclient.com/smartgwt/showcase/.#xpath_xml_integration_category
http://www.smartclient.com/smartgwt/showcase/#json_integration_category_simple

- RestDataSource class
    - per operation dataURLs
    - support for meta data (start row, end row, etc.)
    - Documented format for server inbound data and responses

Example:
http://www.smartclient.com/smartgwt/showcase/#restfulds_xml_integration_category

- 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 ds/test_data/animals.data.xml into a new subdirectory of your project's webroot (war) directory (currently exists under the 'ds' directory of the showcase)
- 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

#7 Data Integration (Continued)

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:
http://www.smartclient.com/smartgwtee/showcase/#simple_custom_ds
Note - extensible format (example "mapped bean class"):
http://www.smartclient.com/smartgwtee/showcase/ - orm_ds

DMI dataSource (serverObject + method to invoke)
Example: 'ds-dmi' example in package

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

Example:
http://www.smartclient.com/smartgwtee/showcase/#scripting_user_specific_data

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.

Example:
http://www.smartclient.com/smartgwtee/showcase/#scripting_validation
- addToTemplateContext (Typically called from an override to the IDACall servlet)

#7 Data Integration (Continued)

- 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"

Example:
http://www.smartclient.com/smartgwtee/showcase/ - transactions_queued_md

- 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
  - serverType= "hibernate"

- 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.
- create a ListGrid bound to this new dataSource, showing the data returned by this operation

DETAILED AGENDA

#8 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
  - atomic responses [yes/no]
  - 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
- WebService class

e) Relogin

goal: seamless handling of user's privileges expiring, integrated with standard server authentication

- achieved via special server responses
- options for handling (RPCManager.loginRequired(), resubmit transaction)

#9 Developer Tools

Note: These tools require the "tools" module

a) Developer Console

- Launching the developer console
  Java: SC.showConsole() vs javascript:isc.showConsole()
- Overview of tabs
  - Results tab: SmartClient logs and direct javascript eval
  - Watch tab (application structure)
  - RPCManager tab – view server turnarounds
- The DOM inspector
- Running code from the Eval area
- Logging Categories and priorities
- Adding logging messages to applications
- Errors and Stack Traces
- Debugging topic in SmartClient Reference

b) DataSource Wizard / Visual Builder

Note: building application views using VisualBuilder will not generate SGWT Java code at this time. Screens built in XML can be loaded / modified in SGWT.

- dataSource wizard

c) DataSource Console (AKA Admin Console)

- Database configuration
- DataSource import

d) DataSource Generator Wizard

e) Automated testing support

Selenium integration (See "Automated Testing" and "Using Selenium" doc topics), and "selenium" subdirectory of SmartGWT package.

- 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

DETAILED AGENDA

#10 Event Handling

a) SmartClient event model

- SmartClient event types
  - Standard per-component events (click, doubleClick, etc)
  - Component level drag/drop
  - Component level focus / keyboard events
  - component-specific events:
    - ListGrid: click, doubleclick, drag/drop of records
    - FormItem events: changed, focus, blur, icon events
    - Calendar eventChanged
- EventObject - event details (source, etc), canceling events
- EventHandler class
- SmartClient event bubbling
- Page level events
  - registerKey
  - Native GWT mechanisms for capturing page level mouse events

b) Loading skins

- Loading skin resource files
- Modifying bootstrap HTML to load skin

c) Anatomy of a skin directory

- load_skin.js
- skin_styles.css
- images directory

c) Creating a custom skin

- start with existing skin
- component images and styles
- loading your custom skin

  See “skinning” topic in documentation

#11 Mobile Development

(See “mobile development” documentation topic)

a) Adaptive or “mobile-focused” components

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

b) Touch Event handling

c) Detecting mobile views

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

d) Remote debugging

e) Packaging SmartClient applications as native apps.

DETAILED AGENDA

#12 Customizing Appearance, Look and Feel (Skinning / Branding)

a) Localization

- GWT has a standard localization mechanism (See GWT live documentation)
- inherit i18n gwt module and enable locales you want:
  - <inherits name='com.google.gwt.18n.I18N'/>
  - <extend-property name="locale" values="en"/>
  - <extend-property name="locale" values="de" />
- Switch to a locale via parameter on URL: ?locale=de, or via a meta data tag:
  - <meta name="gwt:property" content="locale=de" >

SmartGWT ships with localized system messages for a large number of locales.
The above steps will enable this functionality for the locale you selected

To further localize your application:

- for general localization of application-specific messages, follow standard GWT localization techniques:
  - Create Constants and/or Messages interfaces
  - create per-locale .properties files to fulfill those interfaces
  - use GWT.create() to instantiate these within your app and call the APIs to get localized messages
- To override / extend localized system messages,
  - extend SmartGwtMessages
  - create .properties file(s) with the messages you want to modify
  - call i18nUtil.initMessages(...) to make use of your modified system messages
  - Page.getOrientation()

b) Loading skins

- Loading skin resource files
- Modifying bootstrap HTML to load skin

c) Anatomy of a skin directory

- load_skin.js
- skin_styles.css
- images directory

d) Creating a custom skin

- start with existing skin
- component images and styles
- loading your custom skin

See "skinning" topic in documentation

#13 Performance Considerations

a) Simple optimizations

- SmartClient event types
  - Standar
- Understanding create() vs draw() vs show()
- For nested components,
  - avoid drawing children outside parents before adding.
  - consider building hierarchy before draw, where possible
- 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
- quickDrawAheadRatio
  - resultSet.resultSize
  - record component pooling

DETAILED AGENDA

#13 Performance Considerations (Continued)

b) Network Usage

- 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

#14 Optional Modules

- Analytics
- Real-Time Messaging
- Network Performance

#15 Q&A

HELPFUL RESOURCES

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

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

• SmartGwt product homepage: http://www.smartclient.com/smartgwt/
• Online smartgwt showcase: http://www.smartclient.com/smartgwt/showcase/
• Online smartgwtee showcase: http://www.smartclient.com/smartgwtee/showcase/
• Online smartgwt javadocs: http://www.smartclient.com/smartgwt/javadoc/
• SmartGWT and SmartClient nightly builds (LGPL and Evaluation):
• http://www.smartclient.com/builds
• SmartClient and SmartGwt forums: http://forums.smartclient.com/
• Contacting SmartClient: support@isomorphic.com or
http://www.smartclient.com/company/contact.jsp
• SmartClient blog: http://blog.smartclient.com/
• SmartGwt project homepage: http://code.google.com/p/smartgwt/
• includes download links, documentation links, lgpl source code
• SmartClient documentation: http://www.smartclient.com/product/documentation.jsp

SmartGwtEE package

• index.html
• Javadocs for SmartGwt and SmartClient server
• Numerous samples including complete showcase
• per-sample readme files with build instructions

Register Now

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

Smart GWT Training

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