Lesson 4: Building Powerful Multi-tier, Multi-device Applications using DataSnap REST/JSON

David Intersimone "David I"
Vice President of Developer Relations and Chief Evangelist
davidi@embarcadero.com

Mobile App Development

- Lesson 1 – Hello World! My First Multi-Device App
- Lesson 2 – Turning up the Style and Data!
- Lesson 3 – Accessing Local Storage and Databases
- **Lesson 4 – Building Multi-tier, Multi-device Apps using DataSnap REST/JSON**
- Lesson 5 – Connecting Mobile and Desktop using Tethering
- Lesson 6 – Accessing REST and BaaS Cloud Services

Replay links and lesson slides will appear on my blog
http://blogs.embarcadero.com/davidi/

Lesson 4 Agenda

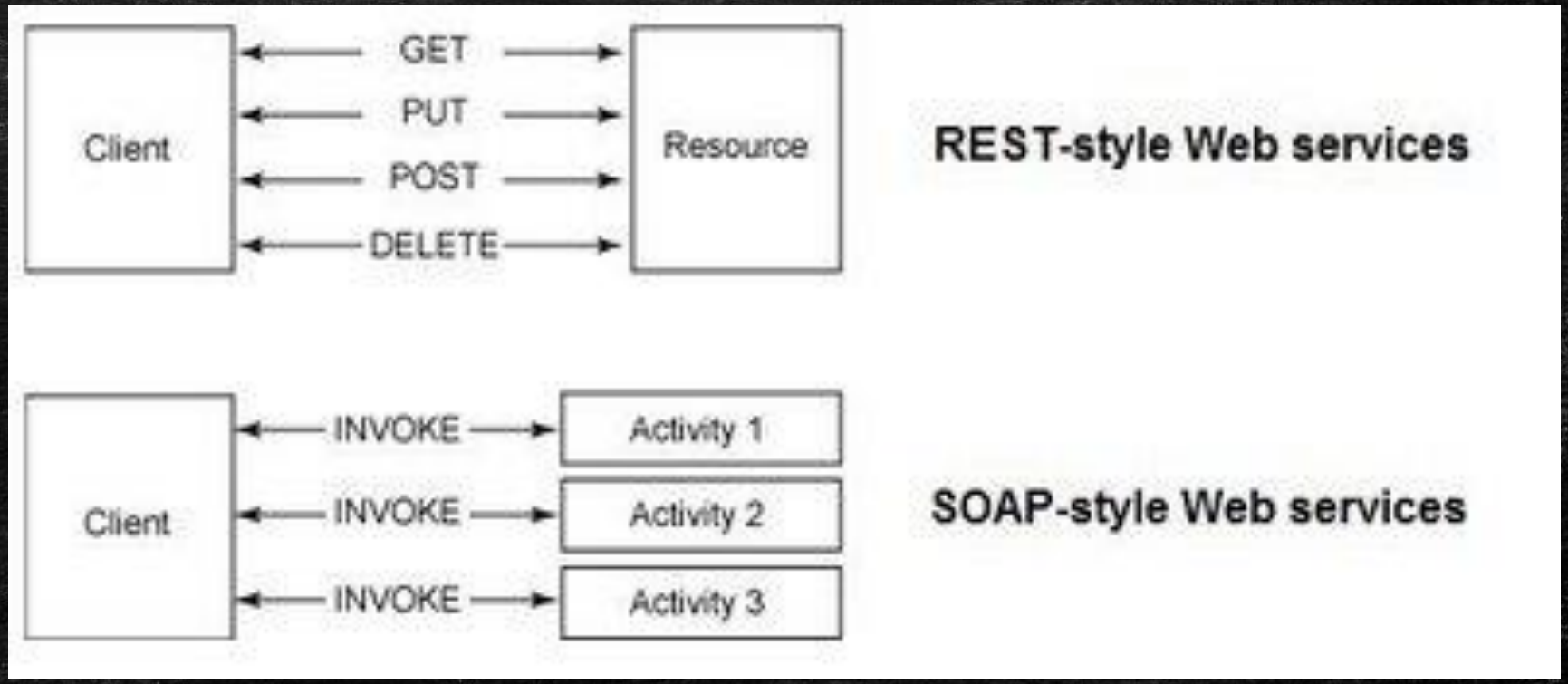
- Why Multi-Tier?
- DataSnap
- FireDAC JSON Reflect
- Samples and Snippets
- Continue development of the mobile business app
- Review, Homework and Next Time
- Q&A

Why Multitier?

- Scalability
- High-availability
- Security
- Fault-tolerance
- Monitoring
- Messaging
- Provisioning

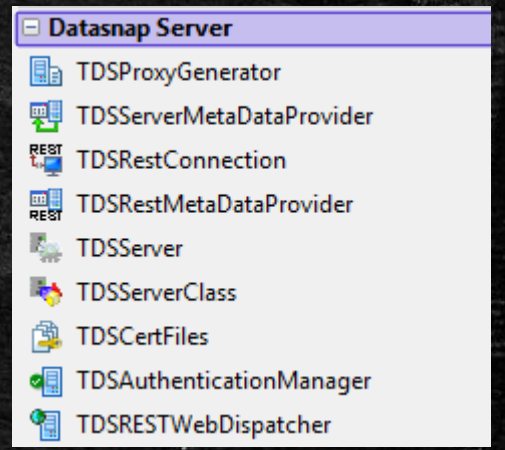
Enterprise Web Services

- Simple: leverages HTTP
- Complex: stack of technologies - XML, XML Schema, SOAP, WSDL, UDDI, …



DataSnap Components

- TDSServer - manages the creation and lifetime of transports and server classes. You need only one TDSServer component per server application.
- TDSServerClass - used to specify a server-side class with published methods that can be called from a remote client using dynamic method invocation. You can have one or more Server classes in your DataSnap server.
  - LifeCycle property
    - **Server**: one instance is created per running server (singleton).
    - **Session**: one instance is created per active client connection.
    - **Invocation**: a new instance is created for each invocation from a client (stateless).
- TDSRESTWebDispatcher
- TDSRestConnection



DataSnap REST Transports

- REST Communication Protocols – HTTP, HTTPS
- Transport Filters
  - Encryption: PC1, RSA
  - Compression: ZLib
  - Custom

Build Scalable Services with DataSnap

- DataSnap wizard
  - File | Other | <language> | DataSnap Server | DataSnap REST Application
  - Project Types
    - Stand-alone application
    - Stand-alone console application
    - ISAPI dynamic link library
    - Apache dynamic link module
  - Protocol and Port
    - Choose your port, test if it is free, find open port
    - Check HTTPS box if you want secure connection – requires you to provide a certificate
- REST Clients and RESTful interfaces using **FireDAC JSON Reflection**
- ApplyUpdates to send back changes

DataSnap Server Methods for a FireDAC Query

Object Pascal

- Method declarations go in the “public” section of the Server Class – Object Pascal
  - function GetDepartmentNames: TFDJSONDataSets;
- Implementation

```
function TServerMethods1.GetDepartmentNames: TFDJSONDataSets;
begin
  // Clear active so that query will reexecute.
  FDQueryDepartmentNames.Active := False;
  Result := TFDJSONDataSets.Create;
  TFDJSONDataSetsWriter.ListAdd(Result, FDQueryDepartmentNames);
end;
```

C++

- Method declarations go in the “public” section of the Server Class in the C++ header file
  - TJSONObject* GetDepartmentNames();
- Implementation

```
TJSONObject* TServerMethods1::GetDepartmentNames()
{
    FDQueryDepartmentNames->Close();
    TFDJSONDataSets *ds = new TFDJSONDataSets();
    TFDJSONDataSetsWriter::ListAdd(ds, FDQueryDepartmentNames);
    TJSONObject *obj = new TJSONObject();
    TFDJSONInterceptor::DataSetsToJSONObject(ds, obj);
    return obj;
}
```

DataSnap Server Method for ApplyUpdates

Object Pascal

```
procedure TServerMethods1.ApplyChangesDepartmentEmployees(
  const ADeltaList: TFDJSONDeltas);
Var
  LApply: IFDJSONDeltasApplyUpdates;
begin
  // Create the apply object
  LApply := TFDJSONDeltasApplyUpdates.Create(ADeltaList);
  // Apply the department delta
  LApply.ApplyUpdates(sDepartment, FDQueryDepartment.Command);
  if LApply.Errors.Count = 0 then
    // If no errors, apply the employee delta
    LApply.ApplyUpdates(sEmployees, FDQueryDepartmentEmployees.Command);
  if LApply.Errors.Count > 0 then
    // Raise an exception if any errors.
    raise Exception.Create(LApply.Errors.Strings.Text);
end;
```

C++

```
void TServerMethods1::ApplyChangesDepartmentEmployees(TJSONObject* AJSONObject)
{
  TFDJSONDeltas *LDeltas = new TFDJSONDeltas();
  TFDJSONInterceptor::JSONObjectToDataSets(AJSONObject, LDeltas);
  TFDJSONErrors *errs = new TFDJSONErrors();
  // Apply the department delta
  TFDJSONDeltasApplyUpdates::ListApplyUpdates(LDeltas, sDepartment,
            FDQueryDepartment->Command, errs);
  // If no errors, apply the employee delta
  if (errs->Count == 0) {
      TFDJSONDeltasApplyUpdates::ListApplyUpdates(LDeltas, sEmployees,
            FDQueryDepartmentEmployees->Command, errs);
  }
  // Raise an exception if any errors.
  if (errs->Count > 0) {
            throw new Exception(errs->Strings->Text);
  }
}
```

DataSnap Client Code to get FireDAC Query

Object Pascal

```
procedure TForm2.GetDepartmentNames;
var
    LDataSetList: TFDJSONDataSets; //need to include
Data.FireDACJSONReflect.
begin
    FDMemTableDepartments.Close;
    // Get dataset list containing Employee names
    LDataSetList :=
ClientModule1.ServerMethods1Client.GetDepartmentNames;
    // Reads the first and only dataset, number 0.
    FDMemTableDepartments.AppendData(
      TFDJSONDataSetsReader.GetListValue(LDataSetList, 0));
    FDMemTableDepartments.Open;
end;
```

C++

```
void TForm2::GetDepartmentNames() {
  TJSONObject* LJSONObject (ClientModule1->ServerMethods1Client-
>GetDepartmentNames());
  std::auto_ptr<TFDJSONDataSets>LDataSets(new TFDJSONDataSets());
  TFDJSONInterceptor::JSONObjectToDataSets(LJSONObject,
LDataSets.get());
  FDMemTableDepartments->Active = false;
  TFDAdaptedDataSet * LDataSet = TFDJSONDataSetsReader::GetListValue
(LDataSets.get(), 0);
  FDMemTableDepartments->AppendData(*LDataSet);
}
```

DataSnap Client ApplyUpdates – Object Pascal

```
function TForm2.GetDeltas: TFDJSONDeltas;
begin
  if FDMemTableDepartment.State in dsEditModes then
  begin
    FDMemTableDepartment.Post;
  end;
  if FDMemTableEmployee.State in dsEditModes then
  begin
    FDMemTableEmployee.Post;
  end;
  // Create a delta list
  Result := TFDJSONDeltas.Create;
  // Add deltas
  TFDJSONDeltasWriter.ListAdd(Result, sEmployees, FDMemTableEmployee);
  TFDJSONDeltasWriter.ListAdd(Result, sDepartment, FDMemTableDepartment);
end;
```

```
procedure TForm2.ApplyUpdates;
var
  LDeltaList: TFDJSONDeltas;
begin
  LDeltaList := GetDeltas;

  // Call server method.  Pass the delta list.

ClientModule1.ServerMethods1Client.ApplyChangesDepartmentEmployees(
LDeltaList);


end;
```

DataSnap Client ApplyUpdates – C++

```
void TForm2::ApplyUpdates()
{
            // Post if editing
            if (dsEditModes.Contains(FDMemTableDepartment->State))
            {
                        FDMemTableDepartment->Post();
            }
            if (dsEditModes.Contains(FDMemTableEmployee->State))
            {
                        FDMemTableEmployee->Post();
            }
            // Create a delta list
            TFDJSONDeltas * LDeltas = new TFDJSONDeltas();
            // Add deltas
            TFDJSONDeltasWriter::ListAdd(LDeltas, sEmployees, FDMemTableEmployee);
            TFDJSONDeltasWriter::ListAdd(LDeltas, sDepartment, FDMemTableDepartment);
            TJSONObject * LJSONObject(new TJSONObject());
            TFDJSONInterceptor::DataSetsToJSONObject(LDeltas, LJSONObject);
            // Call server method.  Pass the delta list.
            ClientModule1->ServerMethods1Client->ApplyChangesDepartmentEmployees(LJSONObject);
 }
```

DataSnap Demo

Next Steps for our Business Mobile App

- Create DataSnap REST Application Server using the
  - Use the local database access components from lesson 3 and put them in a DataSnap REST Application Server Methods Unit
  - Point your FDConnection to InterBase on Windows (or wherever your InterBase server is)
- Run (without debugging) your DataSnap Server and click the start button
- Create a Mobile Client Application
  - Use TDSRestConnection component to connect to you datasnap server
  - Change the code in the Tutorial (it uses Departments and Employees) and use the Customer and Orders queries from lesson 3)
- Run and test your mobile app

Lesson 4 Review

- DataSnap framework makes it easy to create secure, multitier service-oriented architectures
- Multi-level DataSnap Security
  - Transport, Architecture, Deployment
- One codebase for client development on all major mobile and desktop platforms
- IDE wizards and component-based development for ultimate productivity and the fastest time to market

Resources

- DataSnap - Docwiki
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/Developing_DataSnap_Applications
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/DataSnap_REST_Application_Wizard
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/Tutorial:_Using_a_REST_DataSnap_Server_with_an_Application
  - DataSnap Filters Compendium - https://code.google.com/p/dsfc/
- FireDAC - DocWIki
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/FireDAC
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/Overview_(FireDAC)
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/Getting_Started_(FireDAC)
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/Components_(FireDAC)
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/Master-Detail_Relationship_(FireDAC)
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/Data_Explorer
  - http://docwiki.appmethod.com/appmethod/1.14/topics/en/Mobile_Tutorial:_Using_FireDAC_and_SQLite_(iOS_and_Android)
- Videos on Demand
  - C++multi-tier database app with FireDAC JSON Reflection – C++ Mobile Day - http://forms.embarcadero.com/CPPMobileDay6-04
  - Appmethod June 2014 Release in Action - http://www.appmethod.com/june-release
- Blogs
  - http://blogs.embarcadero.com/
  - Pawel Glowacki - C++Builder XE6 multi-tier database app with FireDAC JSON Reflection - http://blogs.embarcadero.com/pawelglowacki/2014/06/04/40330 and his C++ source code project - http://cc.embarcadero.com/item/29887
  - Pawel Glowacki – OpenSSL notes for DataSnap HTTPS - http://blogs.embarcadero.com/pawelglowacki/2013/10/16/40089
  - Marco Cantu – Delphi FireDACJSONReflect for DataSnap - http://blog.marcocantu.com/blog/delphi_xe5_update2_datasnap_firedac.html
  - Pawel Glowacki's Delphi Labs – DataSnap - https://www.embarcadero.com/rad-in-action/delphi-labs

Note: http://docwiki.appmethod.com/appmethod/1.14/topics/en/... = http://docwiki.embarcadero.com/RADStudio/XE6/en/...

Homework & Next Time

- Explore the Docwiki articles and tutorials listed on the Resources page
- Follow the steps in the REST DataSnap server tutorial
- Continue work on the business mobile app
- Lesson 5 – Connecting Mobile and Desktop together using App Tethering
  - The RTL provides **app tethering** components, giving your applications the ability to interact with other applications running either on the same machine or on a remote machine.
    - Discover other applications that are using app tethering, running either on the same device as your application or on other connected devices.
    - Run actions remotely. An application can publish actions using app tethering. Then other applications can remotely invoke any of these actions on the former application.
    - Share data between applications. App tethering allows sharing of standard data types and streams.

Note: http://docwiki.appmethod.com/appmethod/1.14/topics/en/... = http://docwiki.embarcadero.com/RADStudio/XE6/en/...

Q&A

Thank You ☺

davidi@embarcadero.com