QL Integration into Scala and Excel

Martin Dietrich

E.ON Global Commodities

- Over 1000 professionals, active on over 20 exchanges in more than 40 countries
- Over 1000 counterparties in more than 50 countries
- 850,000 trades in 2011
- Market energy, mange commodity risk and provide asset optimization services for the E.ON Group and its third party customers
- Main trading activities: Power, Gas, Emissions, Oil, Coal, Storage
  - Spot, physical forward, options, futures, spread, swaps
  - Swaps, virtual storage, swing gas
  - Physical coal, own fleet of vessels

What makes it special?

- Asset-backed trading
- Permanent obligation to mark and hedge E.ON‘s asset portfolio
- Physical delivery with hundreds of physical constraints in fuel supply and power generation
- Limited liquidity with a significant market share in physical positions
- Simple products like options and forwards
- Complex and structured products like VPP and Swing

Example: Swing Contract

- Periodic delivery within a given delivery period at a given strike price
- Buyer has the right to exercise nomination at short notice (day ahead)
- Min and max number of exercises
- Min and max volume per sub period (month)
- Min and max volume for the whole period (gas year)
- Coupled American style options – flexible but limited exercise
- Complex optimization problems solved by dynamic or linear programming

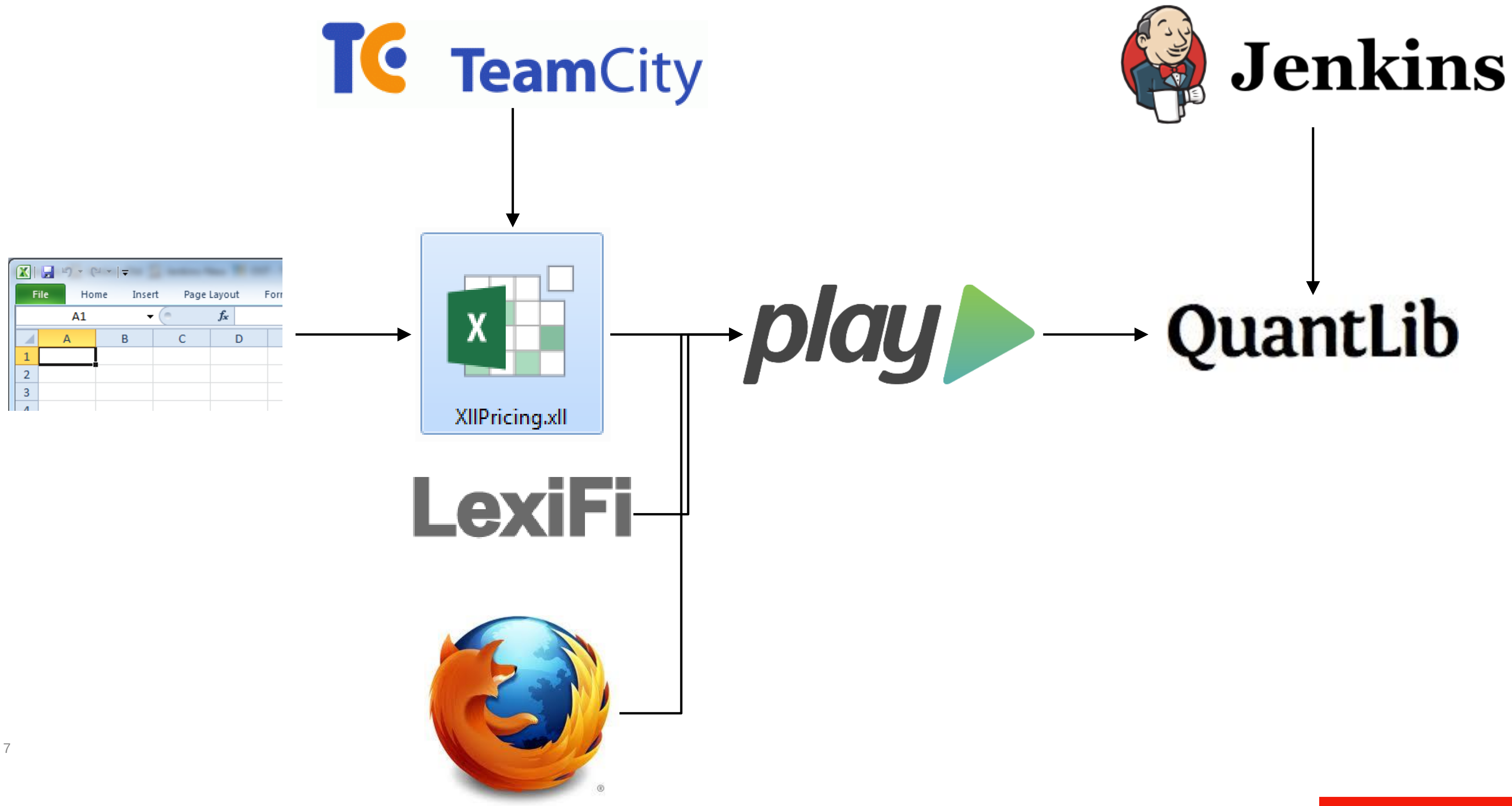
Why QuantLib

- Demand in financial and numerical open source library
- Advanced, mature and tested
- Not reimplementing pricing engines, volatility modelling, Brownian bridge and many more

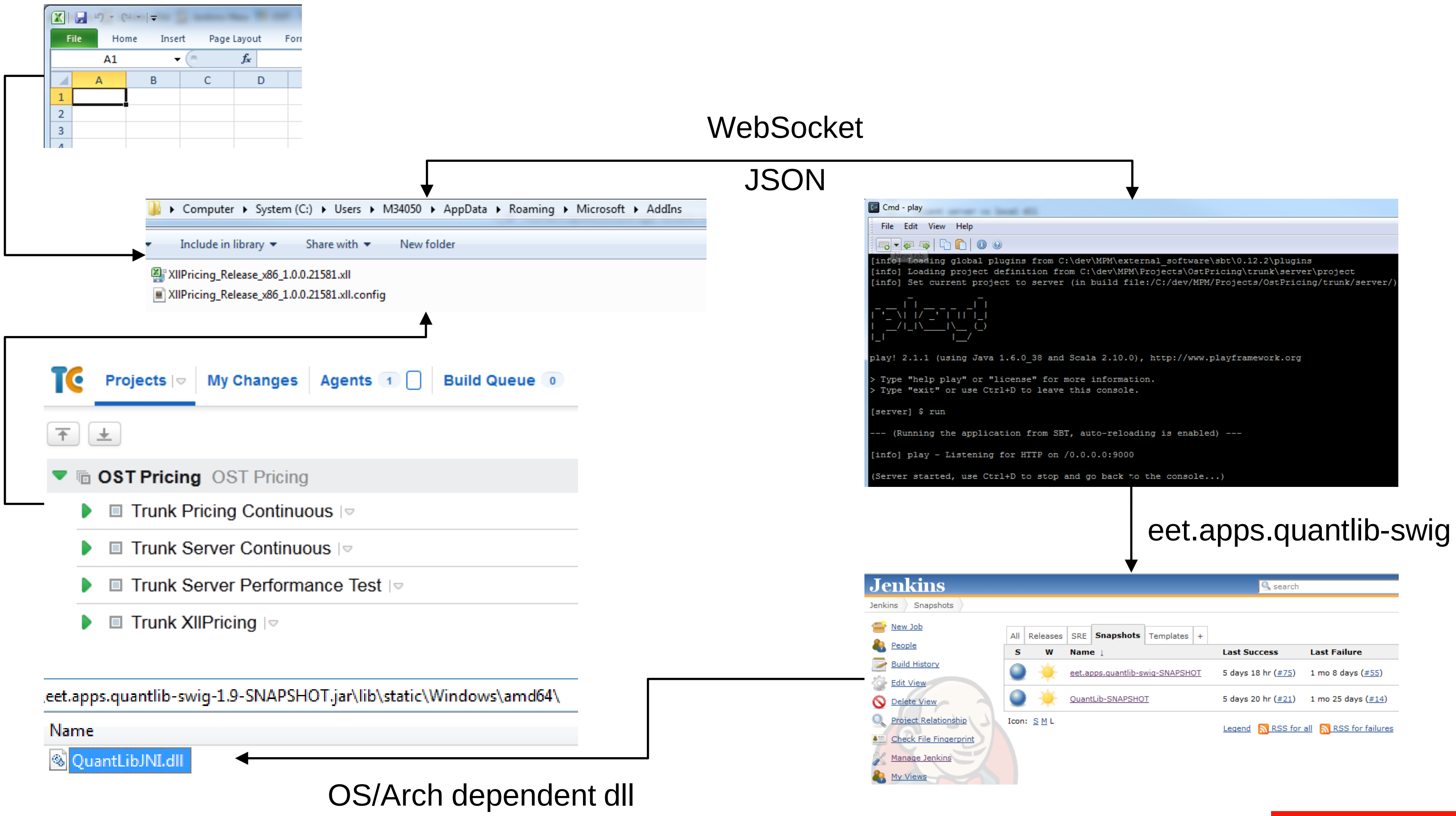
Why not exclusively QuantLib

- Commodity markets are different
- Additional financial engineering requirements
- Want to leverage functional programming languages
- Access identical logic and underlying market data regardless of client
- Big data, half-hourly profiles or forward curves
- Interacting with pricing engines from ETRM, Excel or just a simple browser
- Access the power and performance of a grid from the desktop
- Agile development

Technology Stack



Development Dependencies



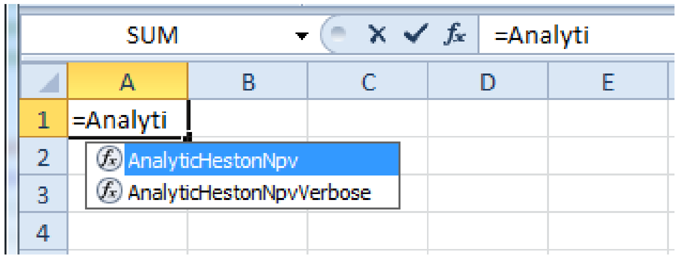
```
[info] Loading global plugins from C:\dev\MPM\external_software\sbt\0.12.2\plugins
[info] Loading project definition from C:\dev\MPM\Projects\OstPricing\trunk\server\project
[info] Set current project to server (in build file:/C:/dev/MPM/Projects/OstPricing/trunk/server/)

play! 2.1.1 (using Java 1.6.0_38 and Scala 2.10.0), http://www.playframework.org

> Type "help play" or "license" for more information.
> Type "exit" or use Ctrl+D to leave this console.

[server] $ run

--- (Running the application from SBT, auto-reloading is enabled) ---

[info] play - Listening for HTTP on /0.0.0.0:9000

(Server started, use Ctrl+D to stop and go back to the console...)
```

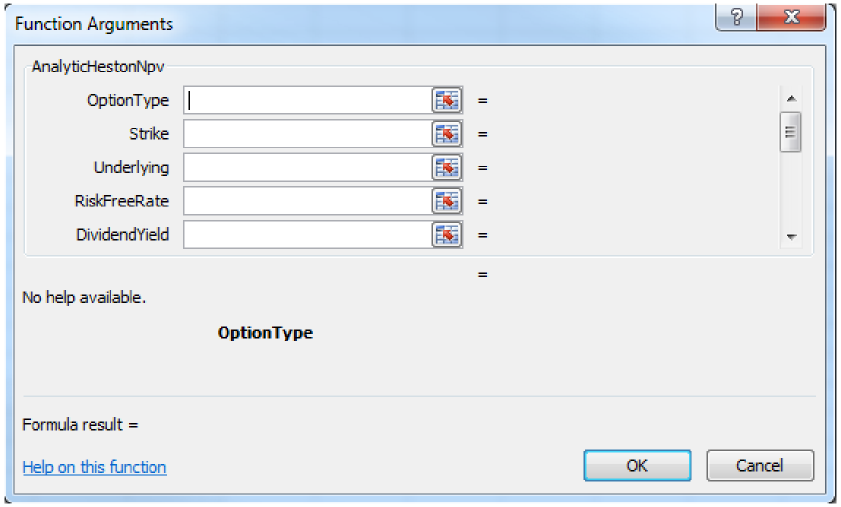
S	W	Name ↓	Last Success	Last Failure
●	☀	eet.apps.quantlib-swig-SNAPSHOT	5 days 18 hr (#75)	1 mo 8 days (#55)
●	☀	QuantLib-SNAPSHOT	5 days 20 hr (#21)	1 mo 25 days (#14)

Icon: S M L

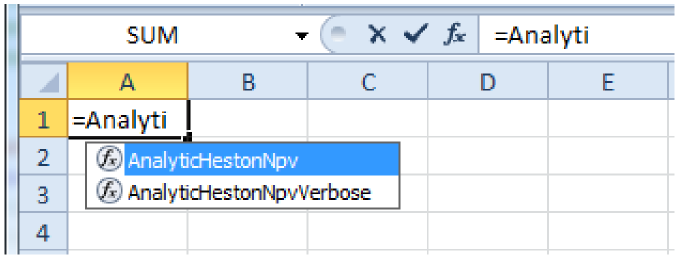
Why Excel-DNA

- Integrating .Net into Excel
- Packaging tool for script files and assemblies to generate a single XLL
- 32/64-bit support
- Asynchronous non-blocking calls
- Task-based operations (.Net 4.0)
- Per-call WebSocket using WebSocket4Net
- Message transfer via JSON using Json.NET
- Automatically resizing the result range



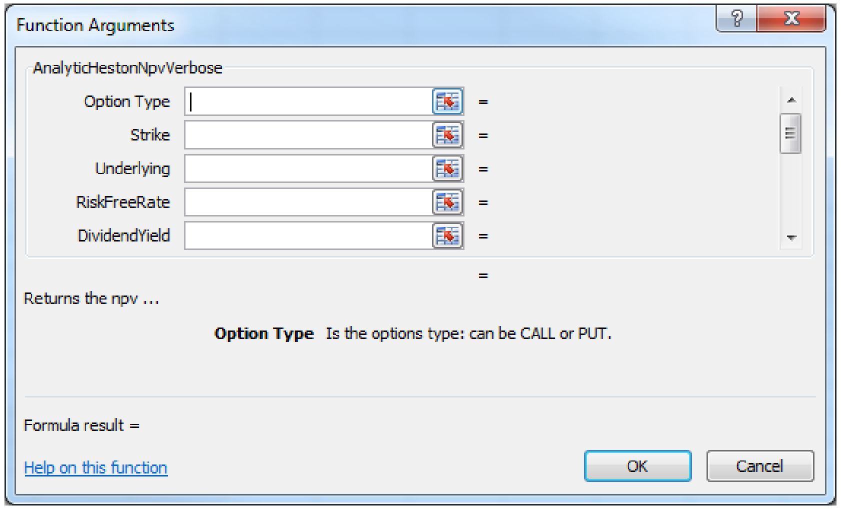


```
public static object AnalyticHestonNpv(
    String optionType, Double strike, ...)
{
    VanillaOption option = new VanillaOption(
        _optionType: optionType,
        _strike: strike, ...);

    return RxExcel.Observe(
        "AnalyticHestonNpv",
        new object[] { optionType, strike, ... },
        () => AnalyticHestonNpvTask(option));
}
```



=Analyti

AnalyticHestonNpv
AnalyticHestonNpvVerbose



Function Arguments

AnalyticHestonNpvVerbose

Option Type =
Strike =
Underlying =
RiskFreeRate =
DividendYield =

=

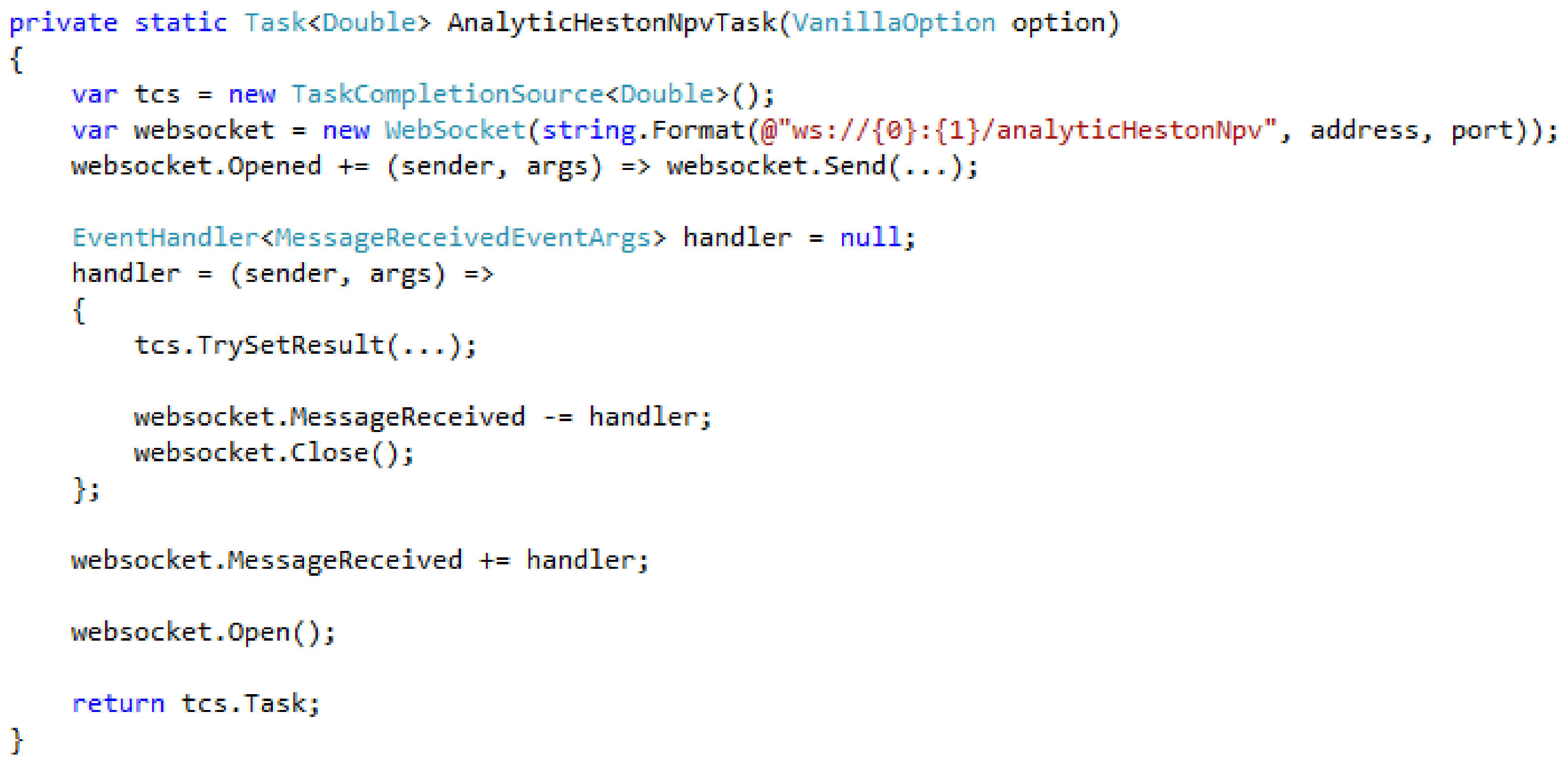
Returns the npv ...

Option Type Is the options type: can be CALL or PUT.

Formula result =

```
[ExcelFunction("Returns the npv ...")]
public static object AnalyticHestonNpvVerbose(
    [ExcelArgument("Is the options type: can be CALL or PUT.",
        Name="Option Type")] String optionType,
    [ExcelArgument("Is the options strike.",
        Name="Strike")] Double strike, ...)
{
    VanillaOption option = new VanillaOption(
        _optionType: optionType,
        _strike: strike, ...);

    return RxExcel.Observe(
        "AnalyticHestonNpv",
        new object[] { optionType, strike, ... },
        () => AnalyticHestonNpvTask(option));
}
```

Interacting with WebSockets



```
private static Task<Double> AnalyticHestonNpvTask(VanillaOption option)
{
    var tcs = new TaskCompletionSource<Double>();
    var websocket = new WebSocket(string.Format(@"ws://{0}:{1}/analyticHestonNpv", address, port));
    websocket.Opened += (sender, args) => websocket.Send(...);

    EventHandler<MessageReceivedEventArgs> handler = null;
    handler = (sender, args) =>
    {
        tcs.TrySetResult(...);

        websocket.MessageReceived -= handler;
        websocket.Close();
    };

    websocket.MessageReceived += handler;

    websocket.Open();

    return tcs.Task;
}
```

Why WebSockets

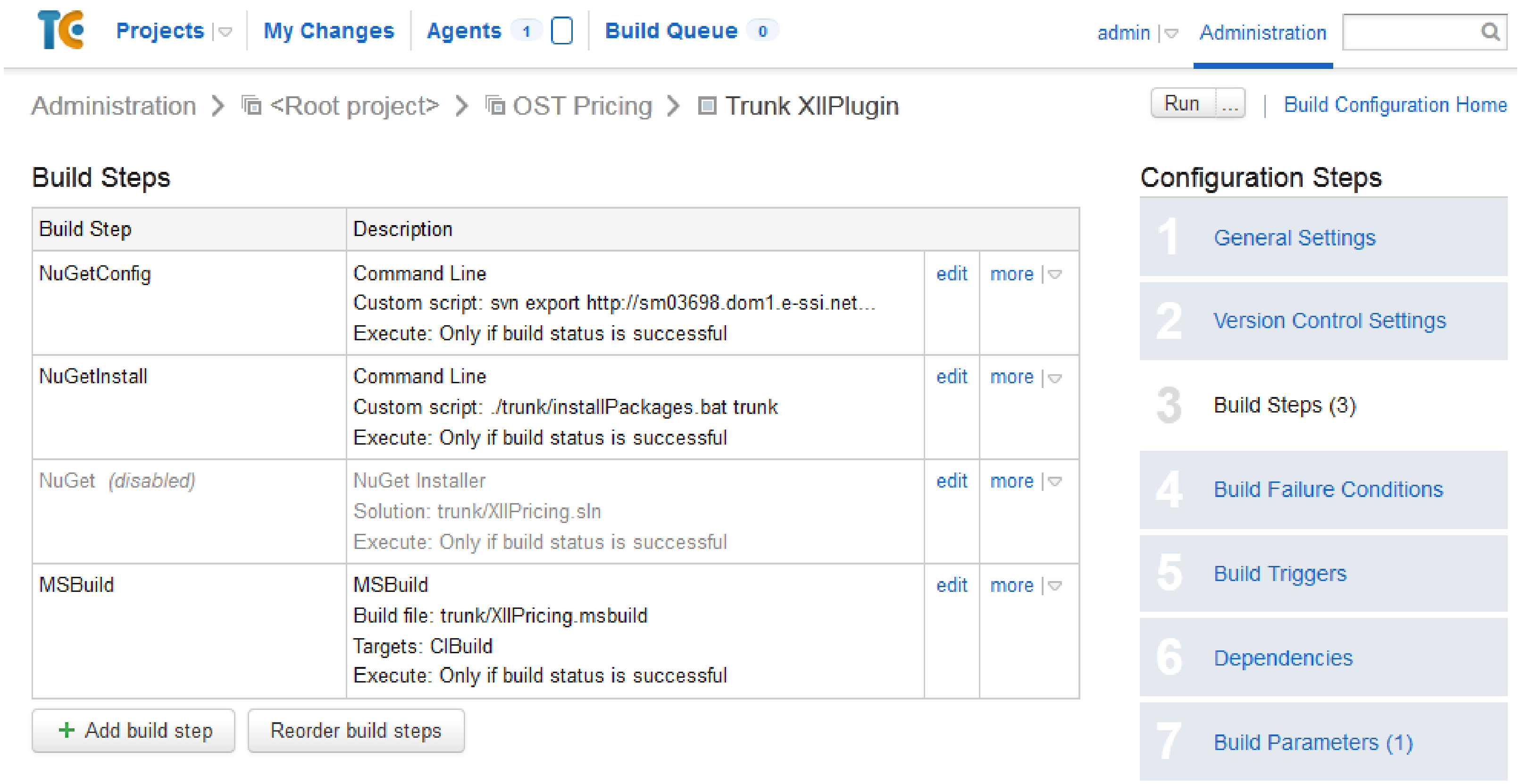
- Stateless protocol
- Real-time full-duplex communication (sending and receiving at a time)
- Alternative to long polling or Comet
- Less bandwith usage
- Initial HTTP request with an upgrade request to the WebSocket protocol
- Independent in and out streams
- No request/response cycle

Why favouring JavaScript Object Notation

- JSON is a text-based data format for data exchange
- Lightwight – no tags, no attributes, less bandwith-intensive
- Limited data types (strings, numerics, Booleans, arrays, objects, nulls)
- Java and .Net APIs at hand for (de)serialization
- Can be persisted in NoSQL databases like MongoDB

```json
{
    "instrument": {
        "exercise": {
            "dates": [
                "2013-09-26T18:00:00"
            ],
            "exerciseType": "European"
        },
        "instrumentCurrency": "EUR",
        "maturity": "2013-09-26T18:00:00",
        "premium": {
            "cashFlows": [
                [
                    "2013-08-21T00:00:00",
                    -15000.0
                ]
            ],
            "currency": "EUR"
        }
    }
}
```

Continuous Integration – the Plugin



Administration > <Root project> > OST Pricing > Trunk XIIPlugin

Build Steps

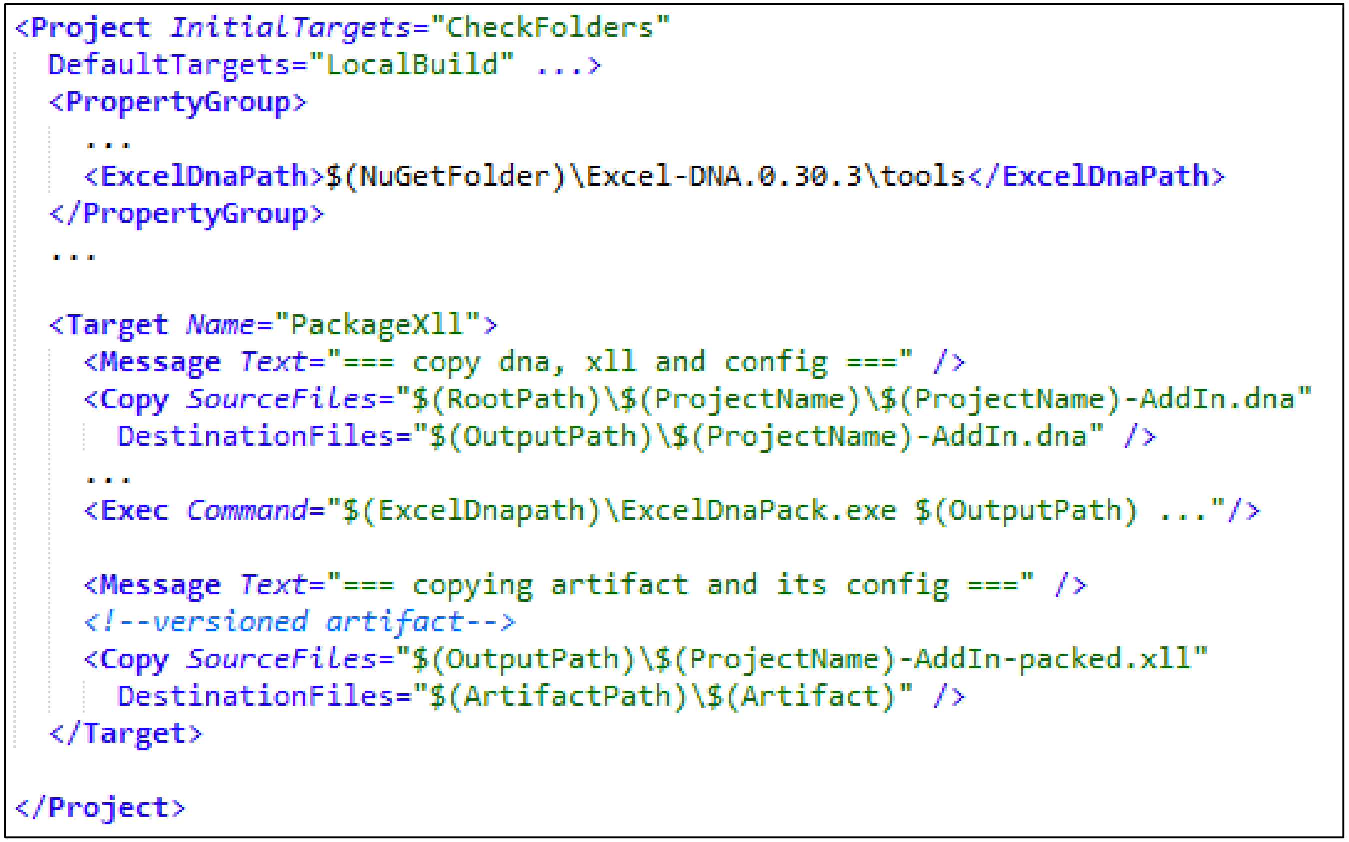
Build Step	Description		
NuGetConfig	Command Line Custom script: svn export http://sm03698.dom1.e-ssi.net... Execute: Only if build status is successful	edit	more
NuGetInstall	Command Line Custom script: ./trunk/installPackages.bat trunk Execute: Only if build status is successful	edit	more
NuGet *(disabled)*	NuGet Installer Solution: trunk/XIIPricing.sln Execute: Only if build status is successful	edit	more
MSBuild	MSBuild Build file: trunk/XIIPricing.msbuild Targets: CIBuild Execute: Only if build status is successful	edit	more

+ Add build step | Reorder build steps

Configuration Steps

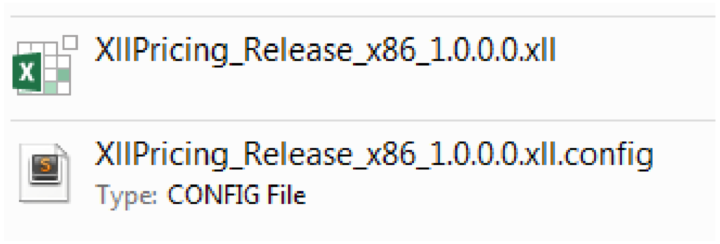
1. General Settings
2. Version Control Settings
3. Build Steps (3)
4. Build Failure Conditions
5. Build Triggers
6. Dependencies
7. Build Parameters (1)

Continuous Integration – the Plugin



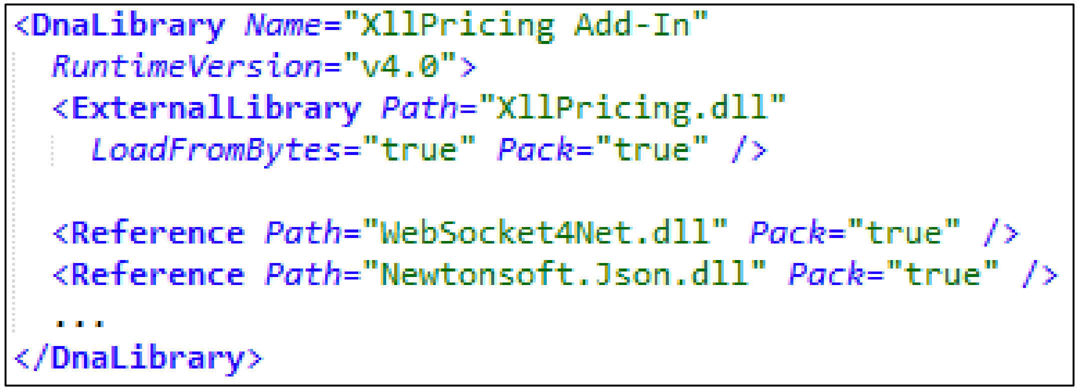
```
<Project InitialTargets="CheckFolders"
  DefaultTargets="LocalBuild" ...>
  <PropertyGroup>
    ...
    <ExcelDnaPath>$(NuGetFolder)\Excel-DNA.0.30.3\tools</ExcelDnaPath>
  </PropertyGroup>
  ...

  <Target Name="PackageXll">
    <Message Text="=== copy dna, xll and config ===" />
    <Copy SourceFiles="$(RootPath)\$(ProjectName)\$(ProjectName)-AddIn.dna"
      DestinationFiles="$(OutputPath)\$(ProjectName)-AddIn.dna" />
    ...
    <Exec Command="$(ExcelDnapath)\ExcelDnaPack.exe $(OutputPath) ..."/>

    <Message Text="=== copying artifact and its config ===" />
    <!--versioned artifact-->
    <Copy SourceFiles="$(OutputPath)\$(ProjectName)-AddIn-packed.xll"
      DestinationFiles="$(ArtifactPath)\$(Artifact)" />
  </Target>

</Project>
```

Continuous Integration – the Plugin

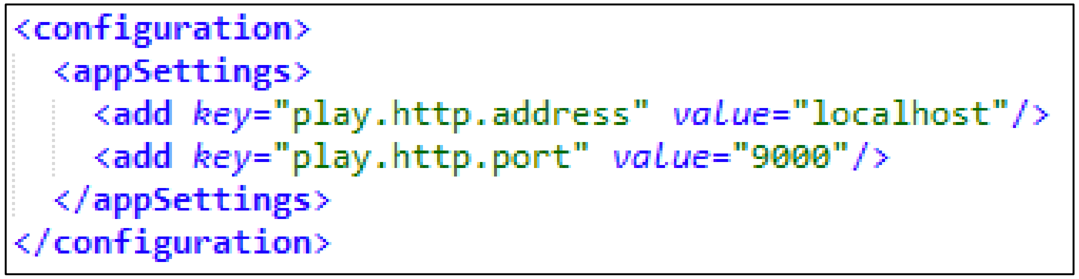


XllPricing_Release_x86_1.0.0.0.xll

XllPricing_Release_x86_1.0.0.0.xll.config
Type: CONFIG File



```
<DnaLibrary Name="XllPricing Add-In"
  RuntimeVersion="v4.0">
  <ExternalLibrary Path="XllPricing.dll"
    LoadFromBytes="true" Pack="true" />

  <Reference Path="WebSocket4Net.dll" Pack="true" />
  <Reference Path="Newtonsoft.Json.dll" Pack="true" />
  ...
</DnaLibrary>
```



```
<configuration>
  <appSettings>
    <add key="play.http.address" value="localhost"/>
    <add key="play.http.port" value="9000"/>
  </appSettings>
</configuration>
```

Why Play

- Full-stack web framework for scala
  - Integrated HTTP server, build system and cache
  - Asynchronous I/O
- Stateless web application
- Live code and configuration changes
- Remote debugging in single threaded environment
- Type safety
- Build-in support for JSON validation
- Build-in support for WebSockets

Exposing a WebSocket with Play

- Specifying the routes

```
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~

# Home page
GET   /analyticHestonNpv  com.eon.pricing.server.Server.analyticHestonNpv
```

- Exposing the WebSocket

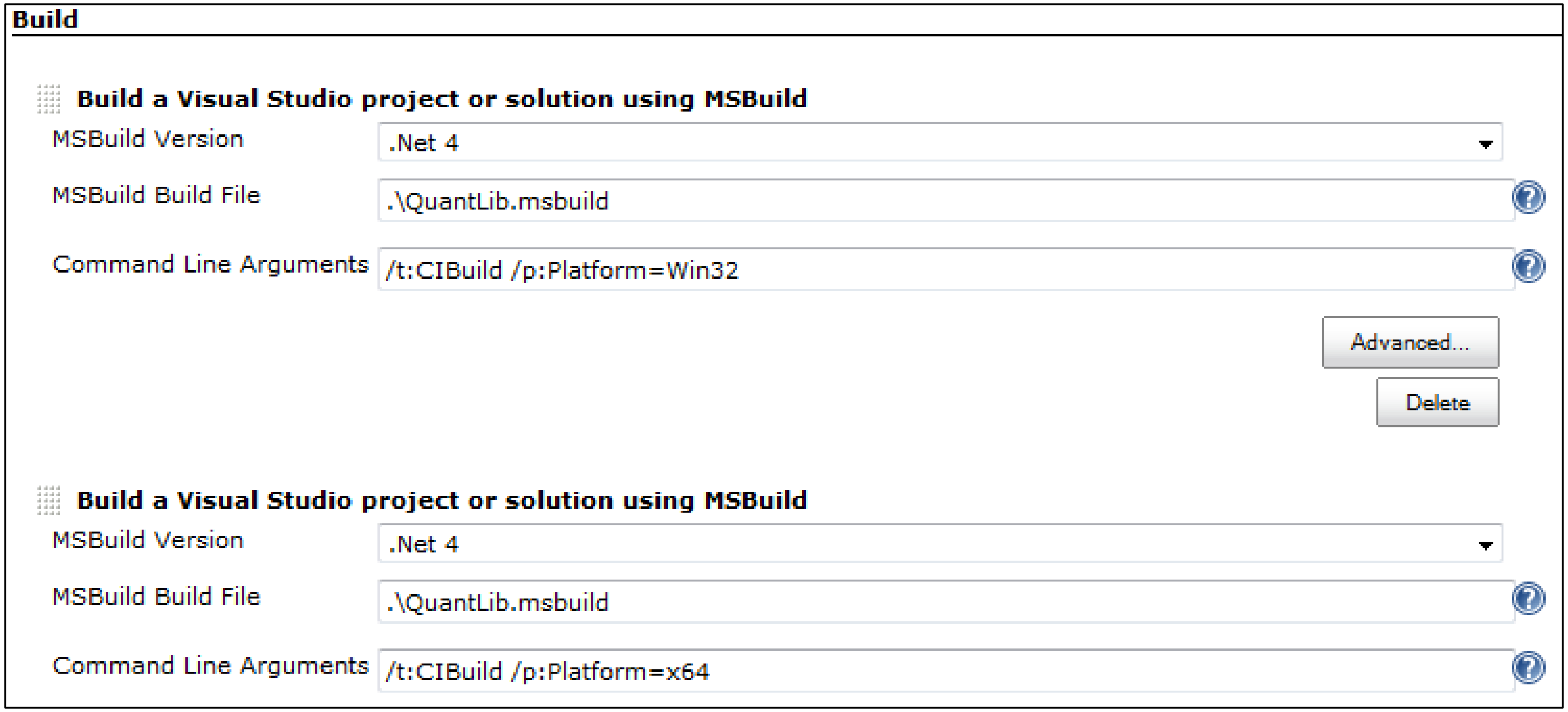
```
object Server extends Controller {

  implicit val simpleFactoryReads = (
    (__ \ "OptionType").read[Option.Type] ~
    (__ \ "Strike").read[Double] ~
    ...
  )(SimpleFactory)

  def analyticHestonNpv = WebSocket.async[JsValue] { request =>
    Akka.future {
      val out = Enumerator.imperative[JsValue]()
      val in = Iteratee.foreach[JsValue] { msg =>
        msg.validate[SimpleFactory] match {
          case JsSuccess(value, _) =>
            val option = new EquityOption(value)
            out.push(
              Json.obj("Value" -> option.analyticHestonNpv))
          ...
        }
      }
      (in, out)
    }
  }
}
```

Exposing QuantLib to Play

- SWIG
  - Simplified Wrapper and Interface Generator
  - Java extension to SWIG writes the Java Native Interface (JNI)
  - SWIG wraps C++ code using Java proxy classes
  - Embedded 32/64bit dll delivered with the jar file, extraction on the fly
    → no need for a separate dll deployment

- QuantLib in a multi-threaded environment
  - SWIG/QuantLib Objects are not shared between different threads
  - Deregister observer during garbage collection via call back hook
  - Thread local singleton pattern

Continuous Integration - QuantLib



Build

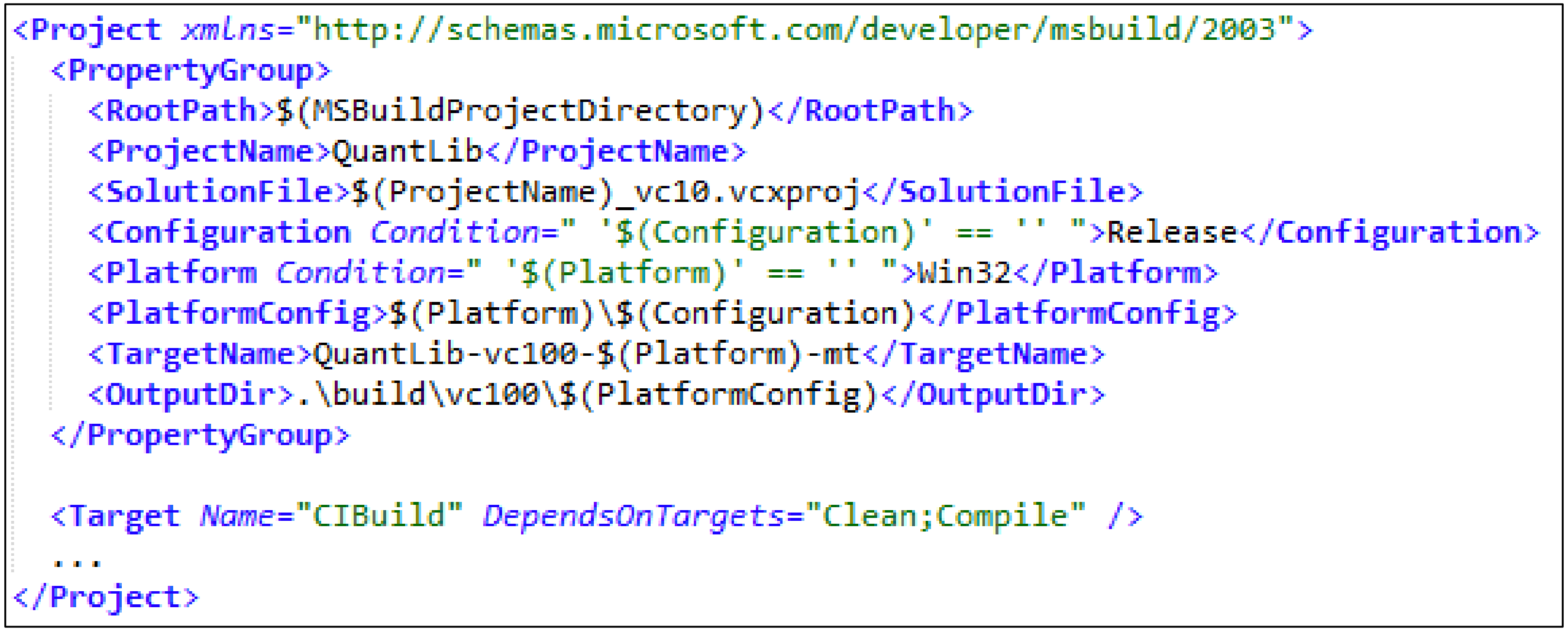
Build a Visual Studio project or solution using MSBuild

MSBuild Version	.Net 4
MSBuild Build File	.\QuantLib.msbuild
Command Line Arguments	/t:CIBuild /p:Platform=Win32

Build a Visual Studio project or solution using MSBuild

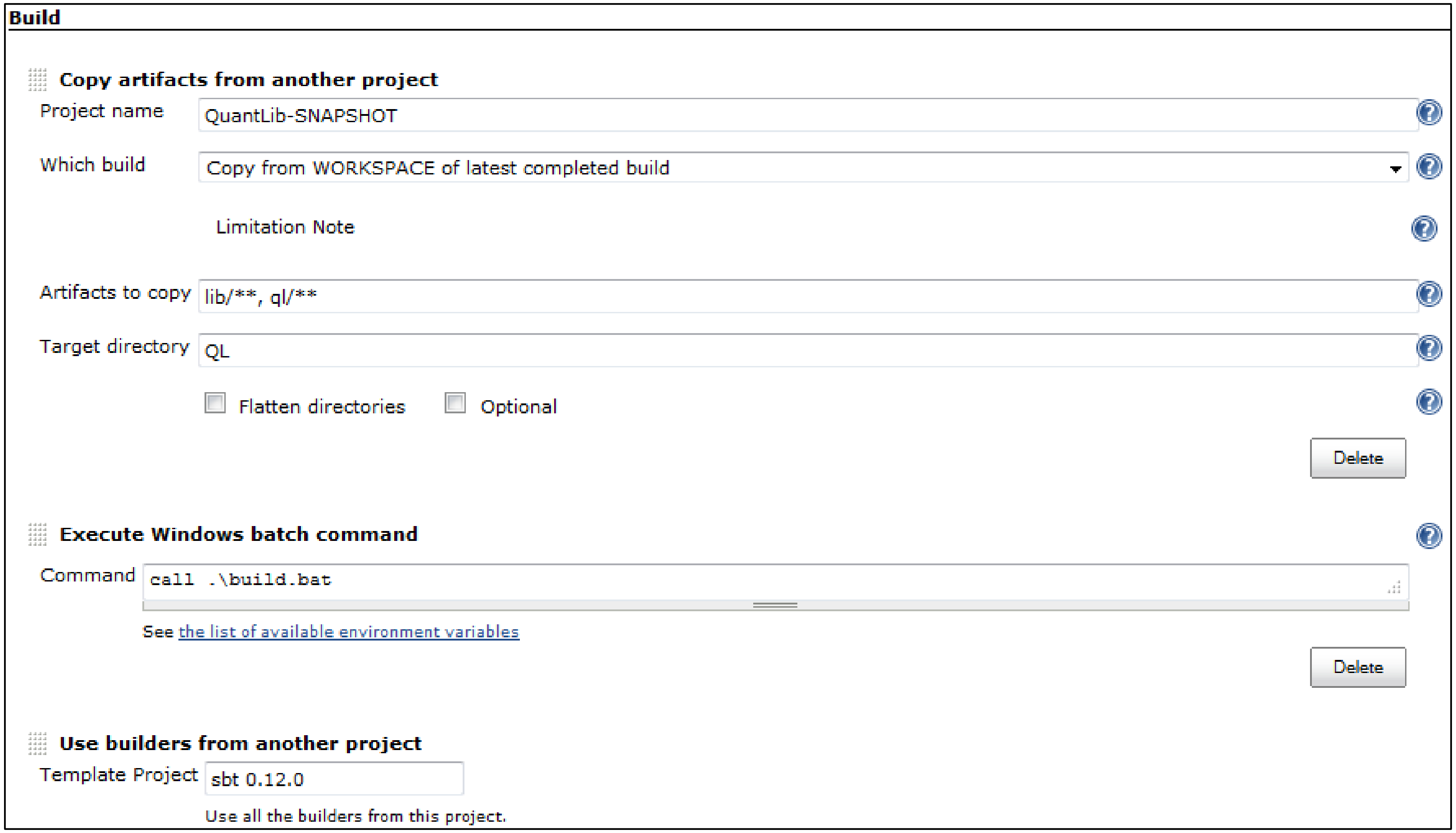
MSBuild Version	.Net 4
MSBuild Build File	.\QuantLib.msbuild
Command Line Arguments	/t:CIBuild /p:Platform=x64

Continuous Integration - QuantLib



```xml
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <RootPath>$(MSBuildProjectDirectory)</RootPath>
    <ProjectName>QuantLib</ProjectName>
    <SolutionFile>$(ProjectName)_vc10.vcxproj</SolutionFile>
    <Configuration Condition=" '$(Configuration)' == '' ">Release</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">Win32</Platform>
    <PlatformConfig>$(Platform)\$(Configuration)</PlatformConfig>
    <TargetName>QuantLib-vc100-$(Platform)-mt</TargetName>
    <OutputDir>.\build\vc100\$(PlatformConfig)</OutputDir>
  </PropertyGroup>

  <Target Name="CIBuild" DependsOnTargets="Clean;Compile" />
  ...
</Project>
```

Continuous Integration - SWIG



Build

Copy artifacts from another project

Project name: QuantLib-SNAPSHOT

Which build: Copy from WORKSPACE of latest completed build

Limitation Note

Artifacts to copy: lib/**, ql/**

Target directory: QL

☐ Flatten directories ☐ Optional

Delete

Execute Windows batch command

Command:

```
call .\build.bat
```

See the list of available environment variables

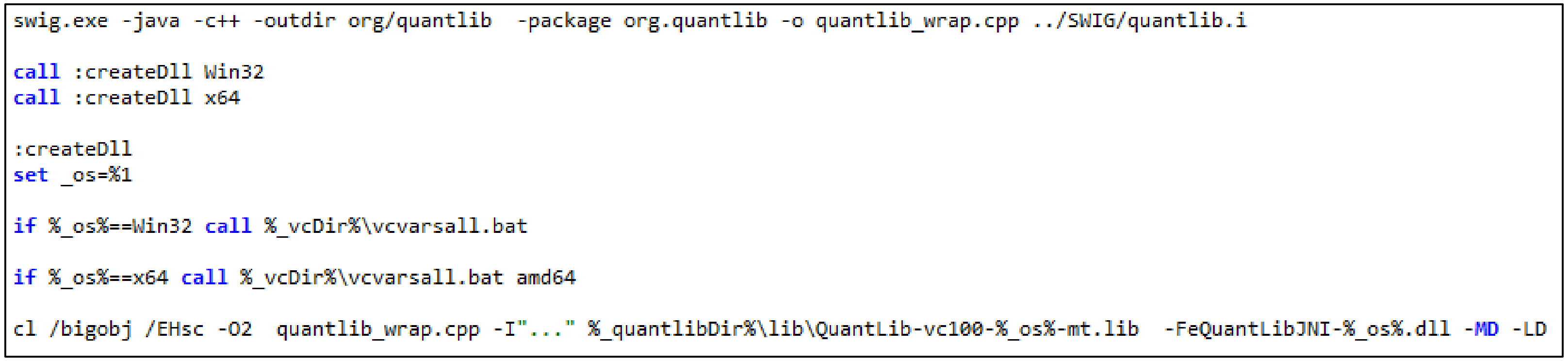
Delete

Use builders from another project

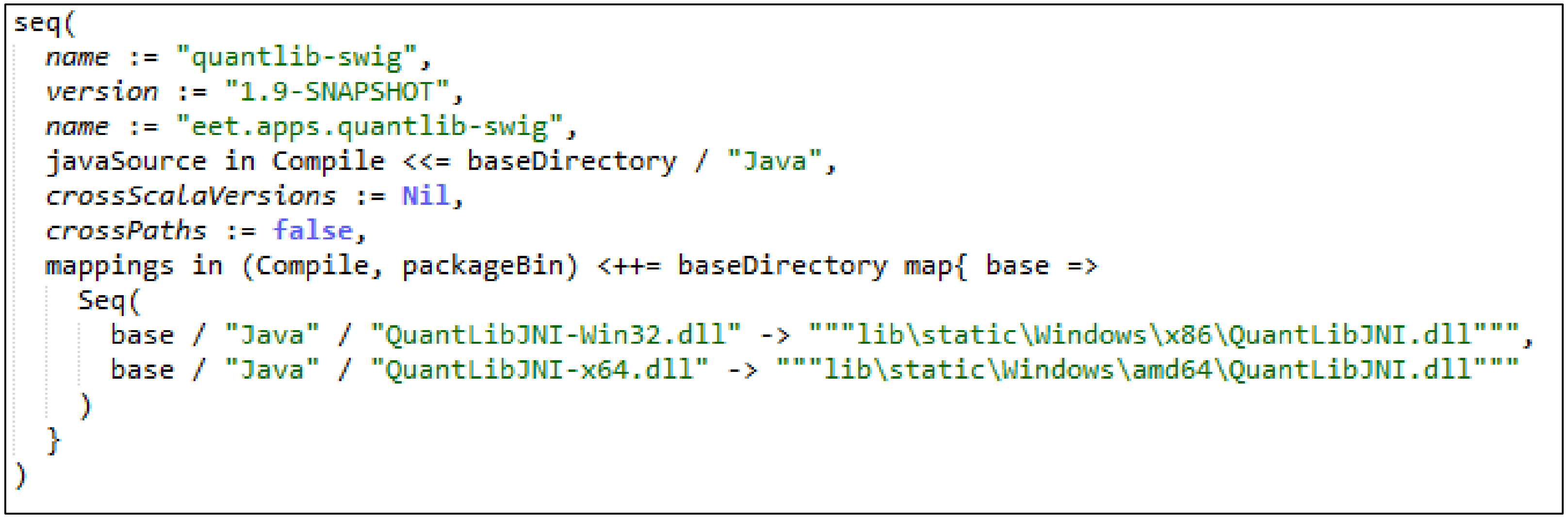
Template Project: sbt 0.12.0

Use all the builders from this project.

Continuous Integration - SWIG



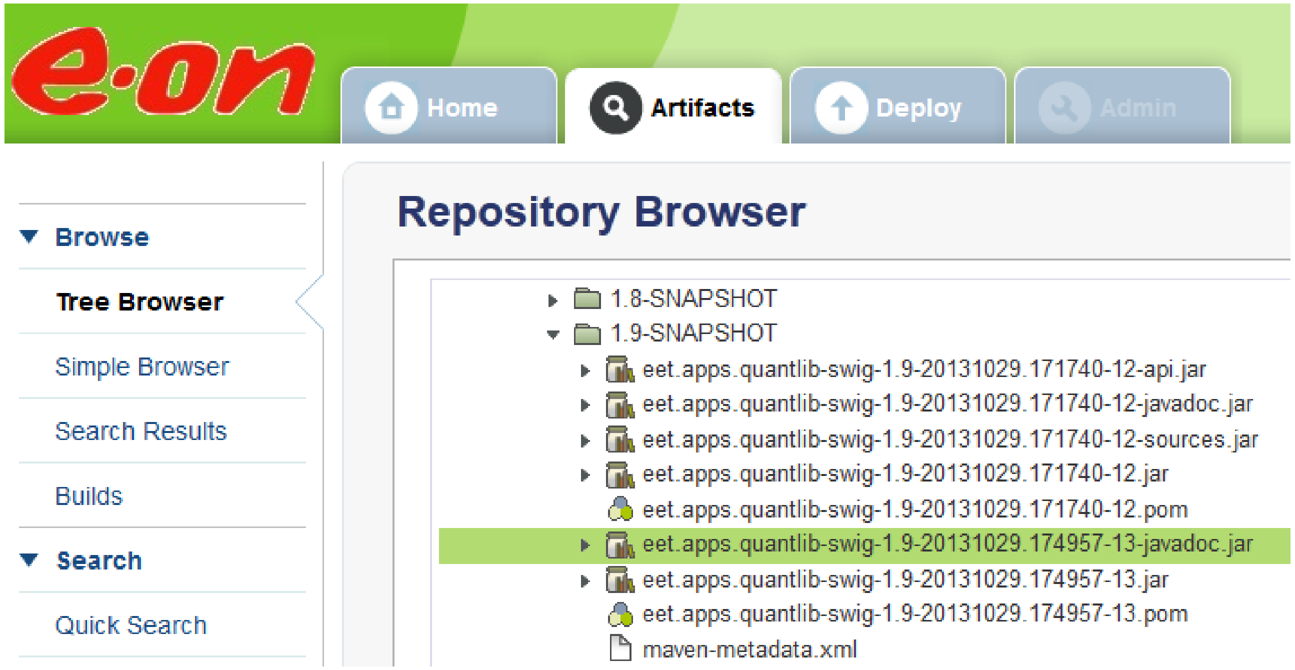
```
swig.exe -java -c++ -outdir org/quantlib  -package org.quantlib -o quantlib_wrap.cpp ../SWIG/quantlib.i

call :createDll Win32
call :createDll x64

:createDll
set _os=%1

if %_os%==Win32 call %_vcDir%\vcvarsall.bat

if %_os%==x64 call %_vcDir%\vcvarsall.bat amd64

cl /bigobj /EHsc -O2  quantlib_wrap.cpp -I"..." %_quantlibDir%\lib\QuantLib-vc100-%_os%-mt.lib  -FeQuantLibJNI-%_os%.dll -MD -LD
```



```
seq(
  name := "quantlib-swig",
  version := "1.9-SNAPSHOT",
  name := "eet.apps.quantlib-swig",
  javaSource in Compile <<= baseDirectory / "Java",
  crossScalaVersions := Nil,
  crossPaths := false,
  mappings in (Compile, packageBin) <++= baseDirectory map{ base =>
    Seq(
      base / "Java" / "QuantLibJNI-Win32.dll" -> """lib\static\Windows\x86\QuantLibJNI.dll""",
      base / "Java" / "QuantLibJNI-x64.dll" -> """lib\static\Windows\amd64\QuantLibJNI.dll"""
    )
  }
)
```

Artifactory

- Central artifact repository for local and remote repositories
- Integrates with maven, ivy and NuGet

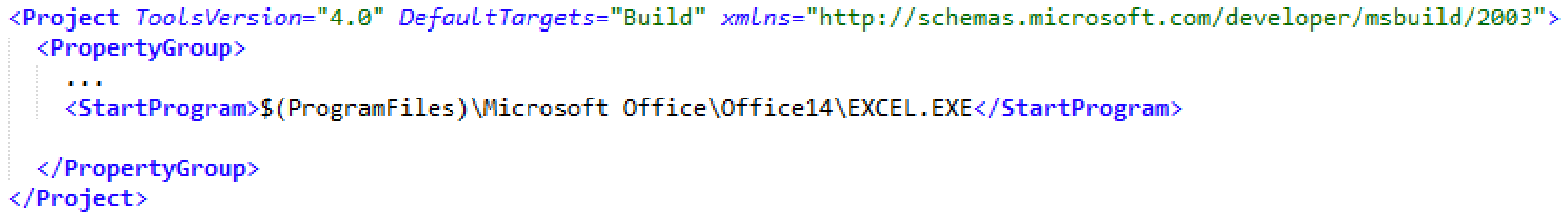


Repository Browser

```
1.8-SNAPSHOT
1.9-SNAPSHOT
  eet.apps.quantlib-swig-1.9-20131029.171740-12-api.jar
  eet.apps.quantlib-swig-1.9-20131029.171740-12-javadoc.jar
  eet.apps.quantlib-swig-1.9-20131029.171740-12-sources.jar
  eet.apps.quantlib-swig-1.9-20131029.171740-12.jar
  eet.apps.quantlib-swig-1.9-20131029.171740-12.pom
  eet.apps.quantlib-swig-1.9-20131029.174957-13-javadoc.jar
  eet.apps.quantlib-swig-1.9-20131029.174957-13.jar
  eet.apps.quantlib-swig-1.9-20131029.174957-13.pom
  maven-metadata.xml
```

Debugging

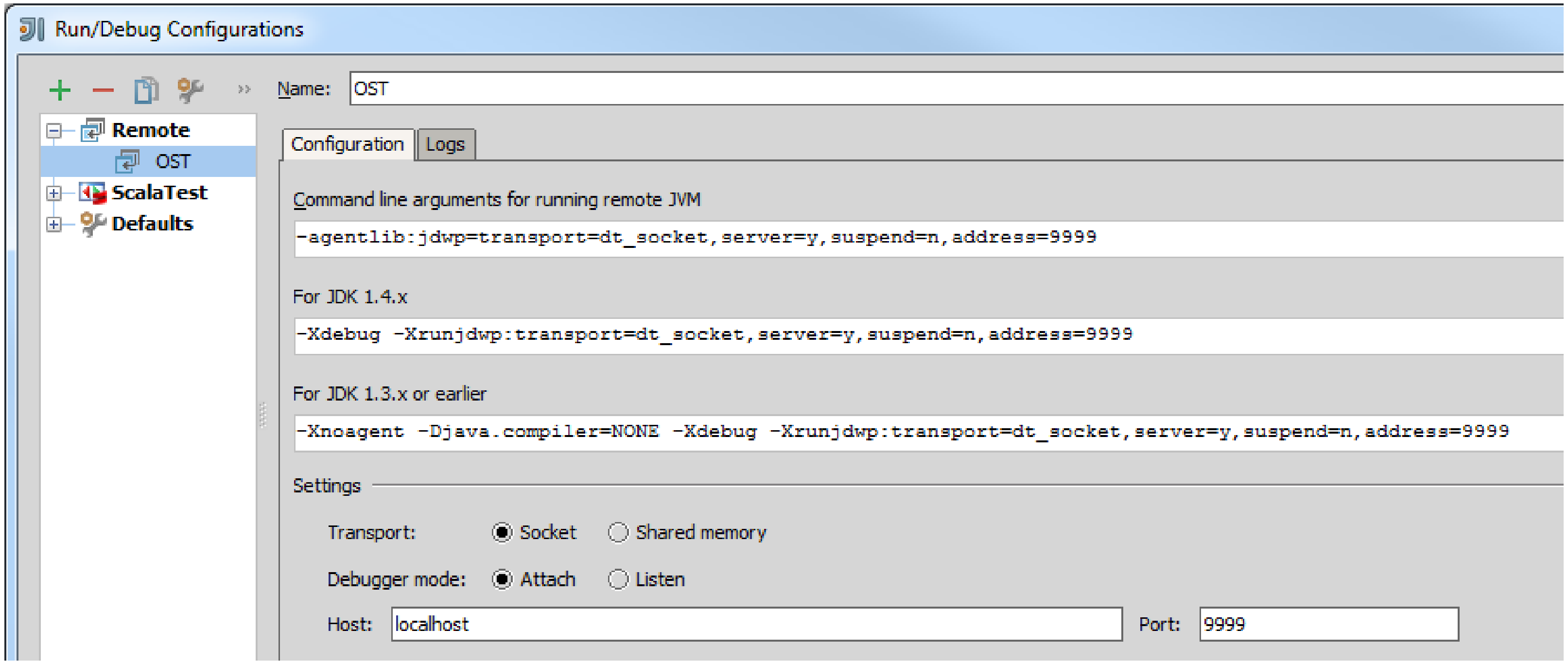
- Start from VS in debug mode - debug your c# code



```xml
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    ...
    <StartProgram>$(ProgramFiles)\Microsoft Office\Office14\EXCEL.EXE</StartProgram>

  </PropertyGroup>
</Project>
```

Debugging

- Run play in debug mode
- Attach remote debugger - debug your scala code



Name: OST

Command line arguments for running remote JVM

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=9999
```

For JDK 1.4.x

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9999
```

For JDK 1.3.x or earlier

```
-Xnoagent -Djava.compiler=NONE -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9999
```

Settings

Transport: [✓ Socket] Shared memory

Debugger mode: [✓ Attach] Listen

Host: localhost Port: 9999

Hands-On

- Pricing a set of vanilla gas options from a spread sheet
- Sending a pricing request from a web browser
- Pricing a vanilla option from LexiFi

Conclusion

- QuantLib can be integrated into multi-language/architechture system
- High throughput
- Scalable with standard web components
- Continous Integraiton and TDD
- Central pricing server

Links and Tutorials

- Principles of Reactive Programming
  https://www.coursera.org/course/reactive

- Functional Programming Principles in Scala
  https://www.coursera.org/course/progfun