MATLAB Production Server Interface for TIBCO Spotfire® Software

Reference Architecture

Contents

Introduction

Spotfire® analytics [1] is a highly visual and open analysis environment that complements packaged applications. A few clicks combine and visualize multi-sourced information with an array of dynamically linked plots that dramatically increase insight and understanding of complex data. Spotfire allows you to capture reusable analysis methods that clearly and easily lead colleagues through approved steps to gather, calculate, visualize and share results.

MATLAB® [2] is the high-level language and interactive environment used by millions of engineers and scientists worldwide. It lets you explore and visualize ideas and collaborate across disciplines including signal and image processing, communications, control systems, and computational finance.

MATLAB Production Server™ [3] lets you run MATLAB® programs within your production systems, enabling you to incorporate custom analytics in enterprise applications.

This reference architecture details the use of MATLAB as the language of technical computing to express advanced analytics and algorithms. The MATLAB algorithms developed in the desktop client can be deployed in a service-oriented architecture (SOA) to address typical production demands handling the scalability and reliability requirements of business-critical applications.

Solutions using such an architecture enables the use of MATLAB's large and powerful collection of statistical and machine learning algorithms and tools for organizing, analyzing and modeling data. The application of this architecture enables a workflow supporting the modeling and visualization of data and analysis right from prototype to full production usage across large enterprises.

System Requirements

This reference architecture is comprised of the following components and was developed using the versions as listed. See the product documentation for any product specific requirements.

MathWorks Products

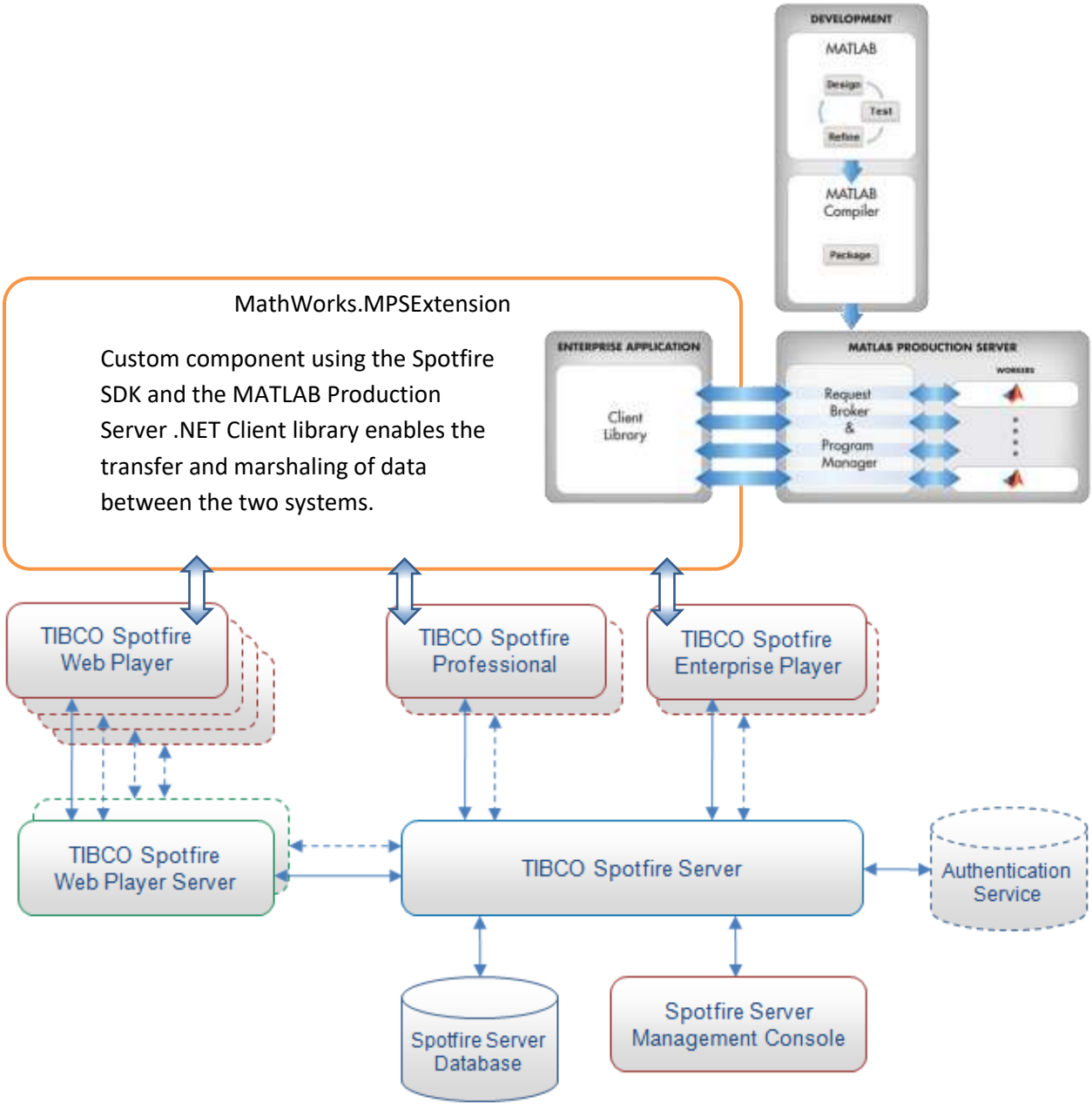
1. MATLAB (R2014b or later)
2. MATLAB Compiler (R2014b or later)
3. MATLAB Production Server (R2014b or later)

TIBCO Products

1. TIBCO Spotfire Server (6.0.0 or later)
2. TIBCO Spotfire Web Player (6.0.0 or later)
3. TIBCO Spotfire Professional or Enterprise (6.0.0 or later)
4. (Optional) TIBCO Spotfire Statistical Services

Reference Diagram

Below is a pictorial architecture of the entire system.



The Spotfire environment provides several interfaces [4] to numerous data sources and information services. In this architecture, MATLAB is used as a computation service enabling the use of MATLAB based analytics with data from a variety of sources. Results of MATLAB analytics are visualized with Spotfire data visualizations and dashboards on the desktop, mobile or within web browsers.

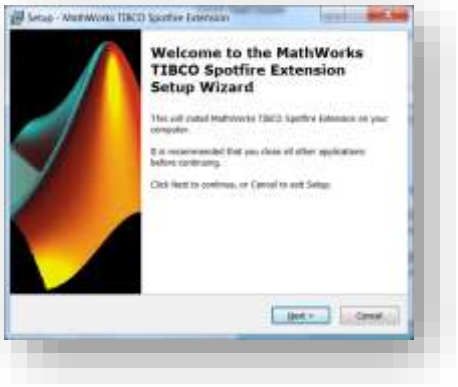
Both Spotfire Server[1] and MATLAB Production Server[2] can be installed in their own standalone or clustered configurations for performance, reliability, fail-over, and disaster recovery. Further discussion on a scaled-out architecture can be found in the Performance Tips in the appendix of this document.

Getting Started Guide

This section is designed to enable the user to quickly set up a demo example integrating TIBCO Spotfire with MATLAB Production Server. Detailed information on Spotfire package deployment, troubleshooting, accessing from mobile applications etc. is available in later sections.

On the machine that TIBCO Spotfire Server is running:

1. Copy the MathWorks_TIBCO_Spotfire_Extension installer to the machine on which TIBCO Spotfire Server is running and then run the installer. This will create two Spotfire package files in the folder; 1) MathWorks.MPSExtension.spk and 2) ProtocolBuffers.spk



Customizing the Configuration

2. Within the MathWorks.MPSExtension.spk file are two XML files that are used to customize the configuration:

   ServerConfiguration.xml is used to configure the URL address for accessing MATLAB Production Server
   FunctionList.xml defines the list of functions running on MATLAB Production Server.

   These files contain information visible to the Spotfire client, and must be updated to contain the correct information. Reference ***Appendix D*** of this document for the process to access the XML files and for rebuild the '.spk' file once the files have been updated.

3. Update the ServerConfiguration.xml follow to define the hostname or IP address and port on which MATLAB Production Server is listening:

[1] Please see: http://stn.spotfire.com/stn/Platform/SpotfireServer.aspx
[2] Please see: http://www.mathworks.com/products/matlab-production-server/features.html#performance-optimization-and-scalability

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfServerListXMLDocument xmlns:xsd="http://www.w3.org/2001/XMLSchema
  <ServerListXMLDocument>
    <hostname>http://localhost</hostname>
    <port>9910</port>
  </ServerListXMLDocument>
</ArrayOfServerListXMLDocument>
```

The FunctionList.xml file lists the MATLAB functions that can be called from Spotfire. This XML file should be modified to include the name of MATLAB function, the input and output parameters, and the data types. This demo uses the optimisePortfolio MATLAB function. The MATLAB code is listed below, followed by the modified FunctionList.xml file.

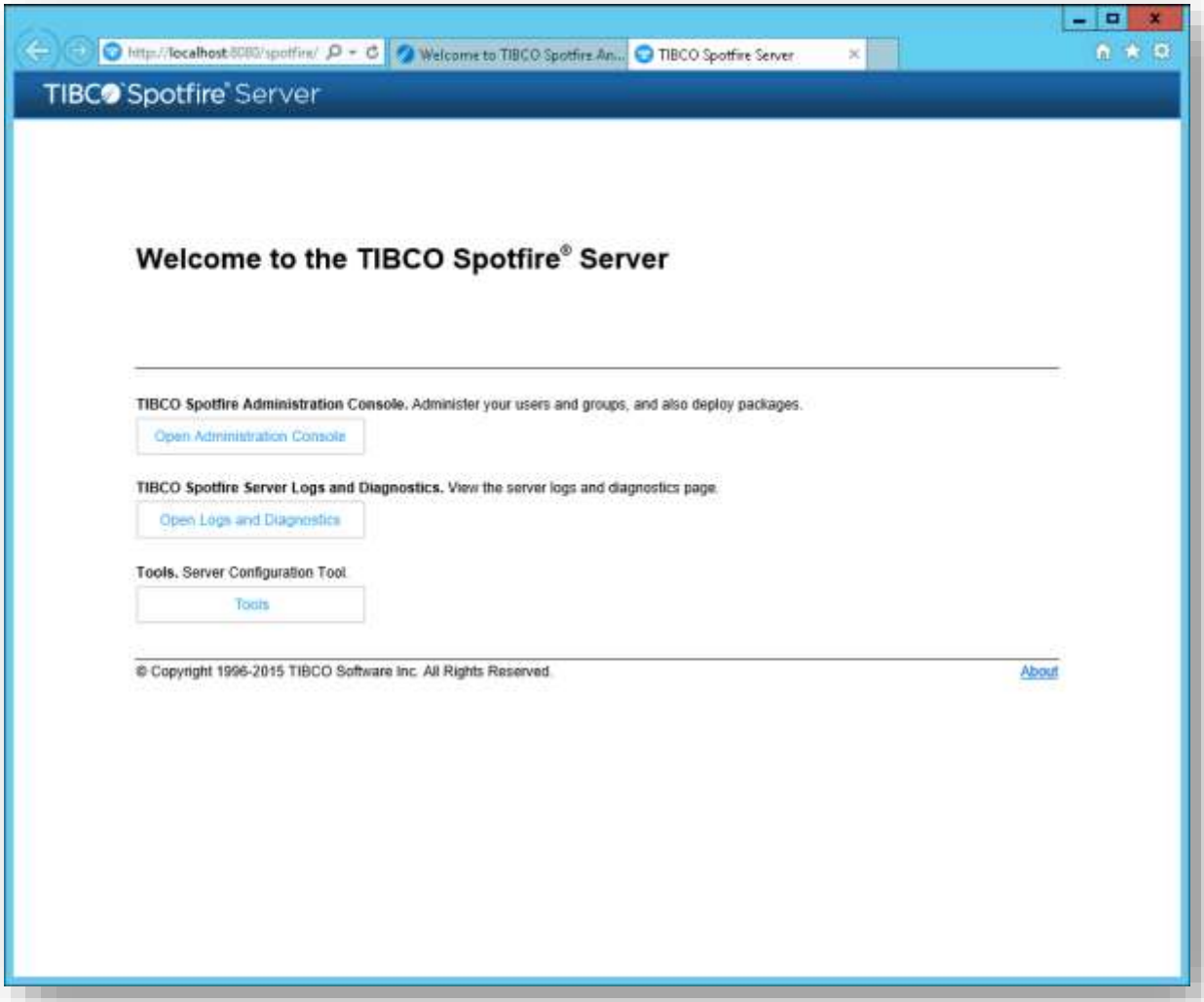
```
function weights = optimisePortfolio(dates, prices, upperBoundWt)
% OPTIMISEPORTFOLIO Simple portfolio optimisation
% This demo computes weights of a portfolio defined by a matrix of asset
% prices for given dates maximising Sharpe ratio;
%
% The portfolio weights are bounded by the same scalar value upperBoundWeight

% Check inputs
dates = dates(:);
validateattributes(prices, {'numeric'}, {'2d'});
validateattributes(upperBoundWt, {'numeric'}, {'scalar'});
assert(size(prices, 1) == length(dates), ...
    'optimisePortfolio: Size mismatch; Expected number of rows in prices is %d and received is %d', ...
    length(dates), size(prices, 1));

% Calculate Returns
% Calculate daily returns
assetReturns = tick2ret(prices, dates, 'Continuous');  % log returns, Financial TB

% Setup portfolio
p = Portfolio;
p = p.estimateAssetMoments(assetReturns); %Mean and covariance of asset returns from return data.
p = p.setDefaultConstraints; %Non-negative weights that must sum to 1.
p = p.setBounds(p.LowerBound, upperBoundWt*ones(1, size(prices, 2)));

% Find min variance portfolio
weights = p.estimateMaxSharpeRatio();
```

```
<FunctionListXMLDocument>
    <archive>portOpt</archive>
    <functionname>optimisePortfolio</functionname>
    <functionsample>System.String[,] weights = optimisePortfolio(System.DateTime[] dates,
System.Double[,] prices, System.Double upperBoundWeight)</functionsample>
    <functionsyntax>optimisePortfolio</functionsyntax>
    <functionhelp>
      <![CDATA[This function computes weights of a portfolio defined by a matrix of asset
prices for
       given dates maximising Sharpe ration; portfolio weights are bounded by the same
scalar value
       upperBoundWeight.<br/>
       The maximization of the Sharpe ratio is accomplished by a one-dimensional
optimization using
       fminbnd to find the portfolio that minimizes the negative of the Sharpe ratio. The
method
       takes only a fully qualified Portfolio object as its input and uses all
information in the
       object to solve the problem.]]>
    </functionhelp>
  </FunctionListXMLDocument>
```

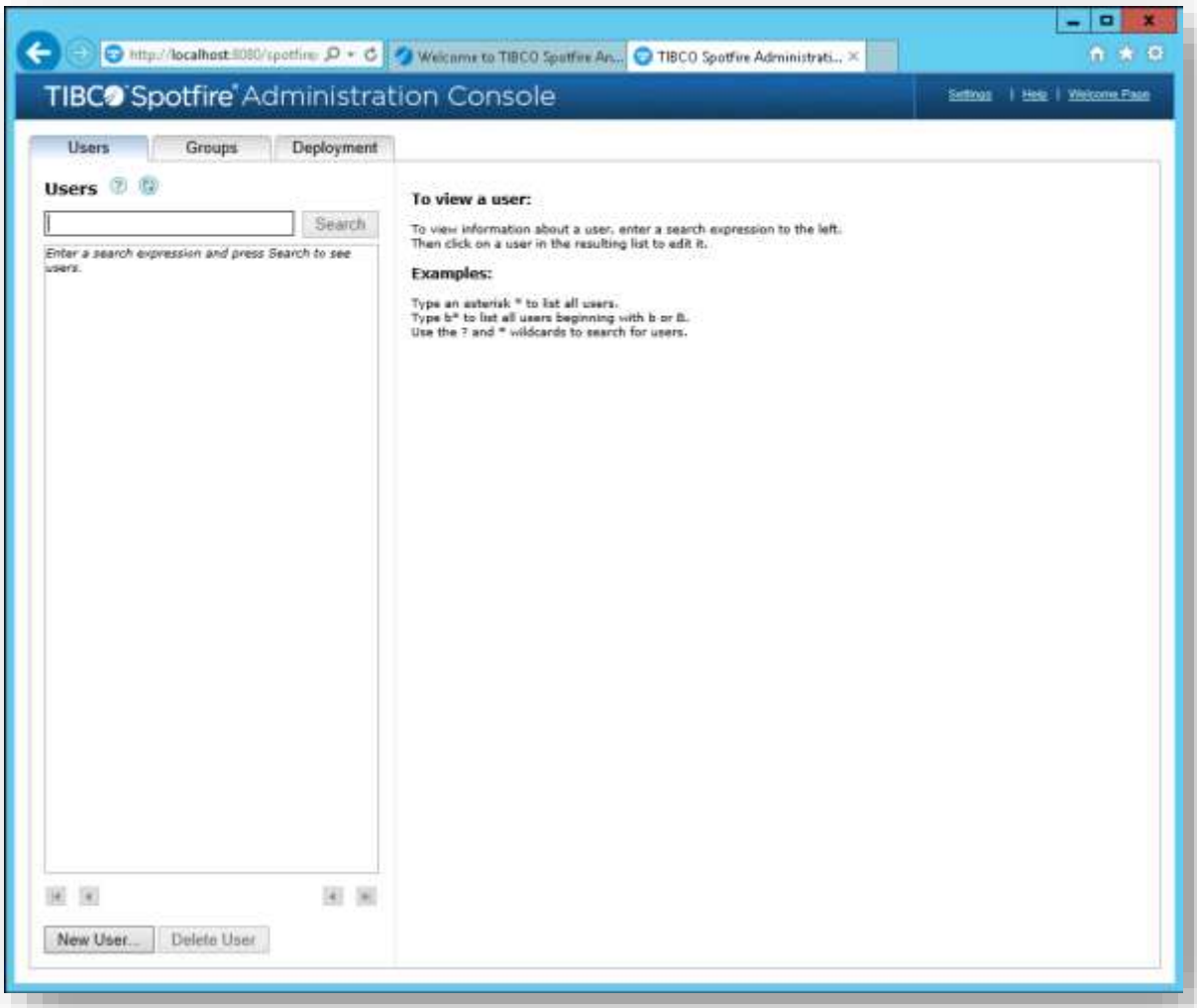
After modifying the two XML files to reflect the MATLAB functions being deployed, follow directions in ***Appendix D*** to recreate the Spotfire package.

Deploying the MATLAB Production Server Interface for TIBCO Spotfire software

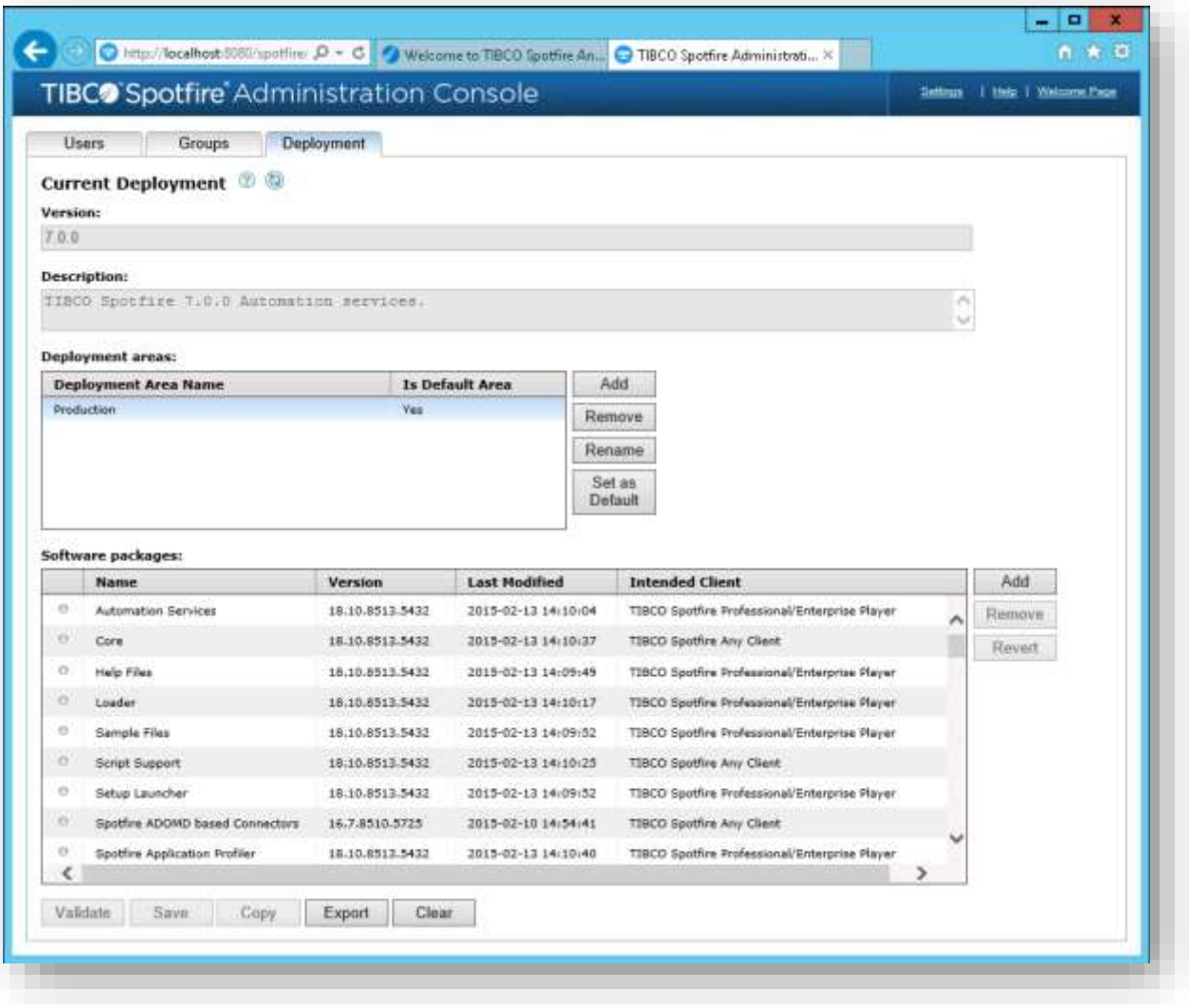
4. To deploy the MATLAB Production Server Interface for TIBCO Spotfire software onto the Spotfire system, open the TIBCO Spotfire Server application.



5. Open the Administration Console.

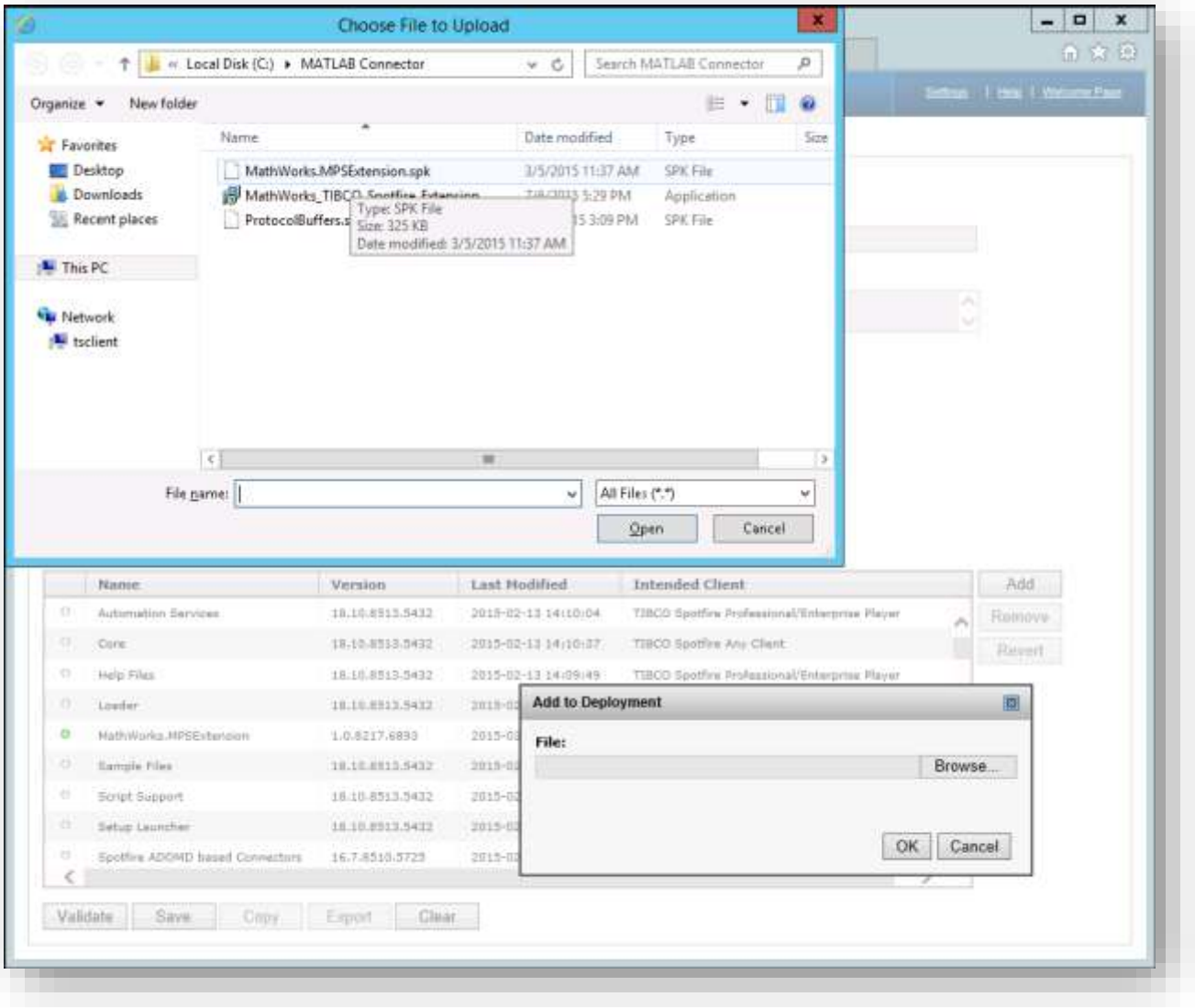


6. Select the Deployment Tab.

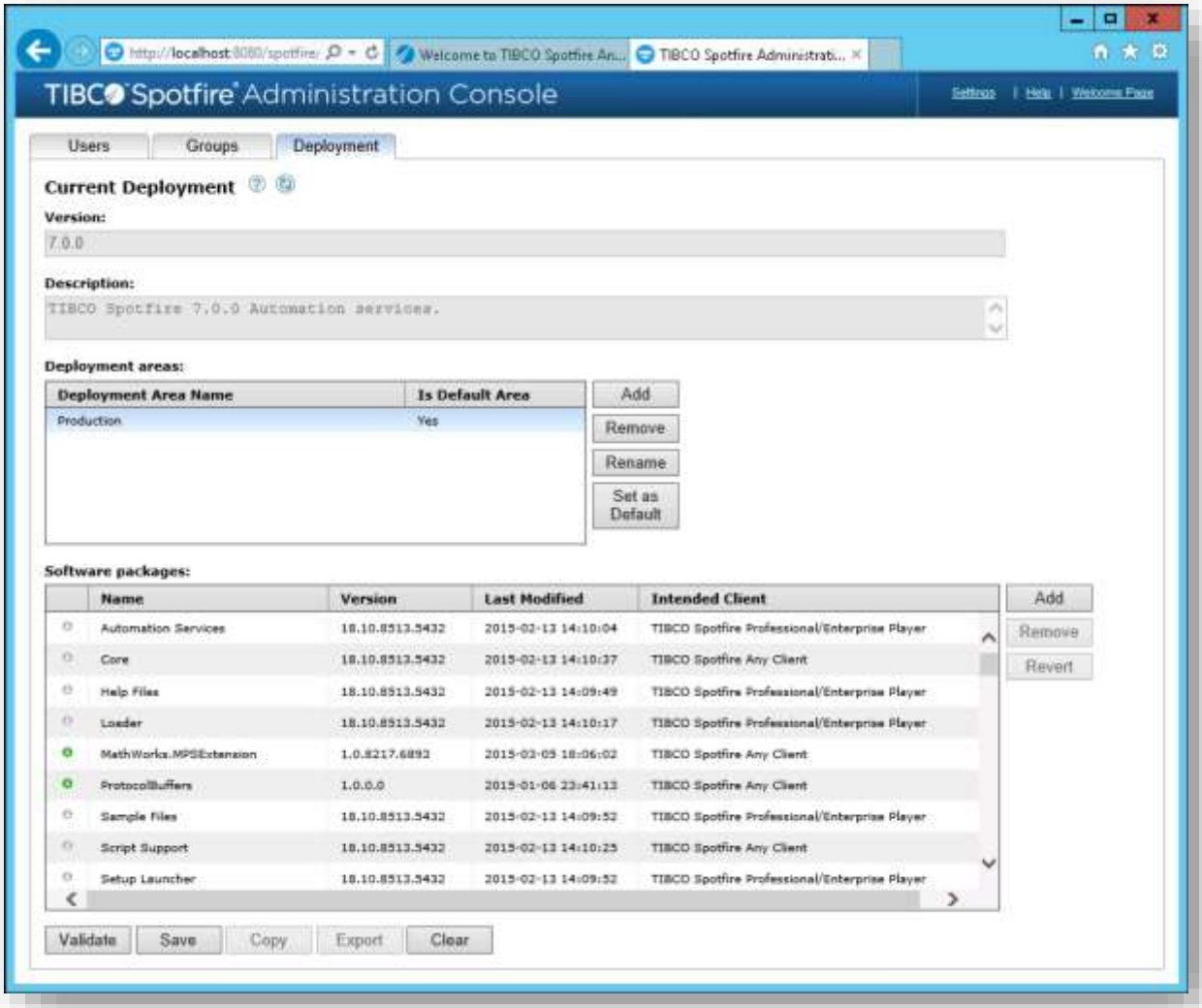


7. Within the “Software Packages” section, use the “Add” button to add the following two files which were extracted from the installer application when it was run:

MathWorks.MPSExtension.spk
ProtocolBuffers.spk



8. Once the two packages are added, select the “Validate” option at the bottom of the page to verify all dependencies. Once you receive the “Validation OK” message, then select “Save”.



Within MATLAB

As an example of how to use the extension, we shall take a simple portfolio optimization demo problem. Portfolio optimization [5] is the process of choosing the proportions of various assets to be held in a portfolio, in such a way as to the portfolio better than any other per some criterion. The criterion will combine, directly or indirectly, considerations of the expected value of the portfolio's rate of return as well as of the return's dispersion and possibly other measures of financial risk.

In this demo, MATLAB is used to compute weights of a portfolio defined by a matrix of asset prices for given dates maximizing the Sharpe ratio [6]. The portfolio weights are bounded by the same scalar value as the upper Bound. The Sharpe ratio characterizes performance of an investment asset in compensating the investor for the risk taken. When comparing two assets versus a common benchmark,

the one with a higher Sharpe ratio provides better return for the same risk (or, equivalently, the same return for lower risk).

This is a single example of using MATLAB as the computation engine and the choice of example could just as easily be rewritten across a number of industries and application areas.

Prototyping Workflow

For the portfolio optimization problem, the function signature looks like:

```
function weights = optimisePortfolio(dates, prices, upperBoundWt)
```

9. The first step in building a robust function is to check that our inputs are valid.

```
%% Check inputs
dates = dates(:);
dates = cell2mat(dates);
prices = cell2mat(prices);
upperBoundWt = cell2mat(upperBoundWt);
validateattributes(prices, {'numeric'}, {'2d'});
validateattributes(upperBoundWt, {'numeric'}, {'scalar'});
assert(size(prices, 1) == length(dates), ...
    'optimisePortfolio: Size mismatch; Expected number of rows in prices
is %d and received is %d', ...
    length(dates), size(prices, 1));
```

Leveraging the MathWorks financial toolbox, MATLAB provides a very compact and readable way to express many algorithms. This is no exception. Our optimization code looks like:

```
%% Calculate Returns
% Calculate daily returns
assetReturns = tick2ret(prices, dates, 'Continuous');  % log returns,
Financial TB

%% Setup portfolio
p = Portfolio;
p = p.estimateAssetMoments(assetReturns); %Mean and covariance of asset
returns from return data.
p = p.setDefaultConstraints; %Non-negative weights that must sum to 1.
%p = p.setBounds(p.LowerBound, upperBoundWt*ones(1, size(prices, 2)));

%% Find min variance portfolio
weights = p.estimateMaxSharpeRatio();
```

Running the code in MATLAB allows users to develop and prototype ideas quickly. Visualization of the data using the MATLAB desktop based graphics system gives quick and early feedback on algorithm designs. For example, to run our function we use asset prices from an XLS spreadsheet. Our data comes from historic daily close prices of global large-cap equity indices, from April 1993 to July 2003.

This dataset contains asset prices from

1. . (TSX) Canadian TSX Composite
2. (CAC) French CAC 40
3. (DAX) German DAX
4. (NIK) Japanese Nikkei 225
5. (FTSE) UK FTSE 100
6. (SP) US S&P 500

A small sample of this data looks like:

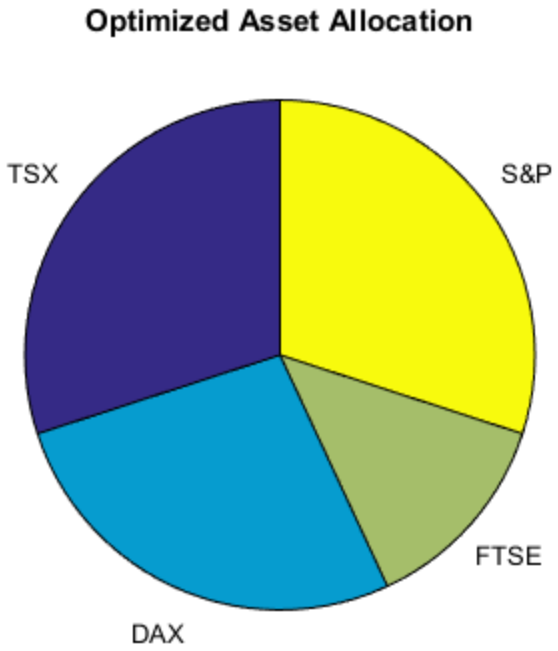
	A	B	C	D	E	F	G
1	**Dates**	**Canada**	**France**	**Germany**	**Japan**	**UK**	**US**
2	4/27/1993	3691.2	1927.4	1640.8	20207	2832.7	438.01
3	4/28/1993	3710.2	1942.5	1628.9	20455	2797.3	438.02
4	4/29/1993	3755	1920.6	1623.9	20687	2786.8	438.89
5	4/30/1993	3789.4	1939	1627.2	20919	2813.1	440.19

Reading in the data and calling the optimization:

```
%% Import Data
% Read asset prices from an Excel spreadsheet
[prices, ~, rawData] = xlsread('IndexData.xlsx');
assetNames = rawData(1, 2:end);
dates = datenum(rawData(2:end, 1), 'dd/mm/yyyy');

%% Set parameters
upperBoundWeight = 0.3;

%% Optimise portfolio
weights = optimisePortfolio(dates, prices, upperBoundWeight);
```

```
%% Display results
f = figure;
pie(weights,{'TSX','CAC','DAX','NIK','FTSE','S&P'});
title('Optimized Asset Allocation');
```

Early and immediate feedback can be visualized in MATLAB.



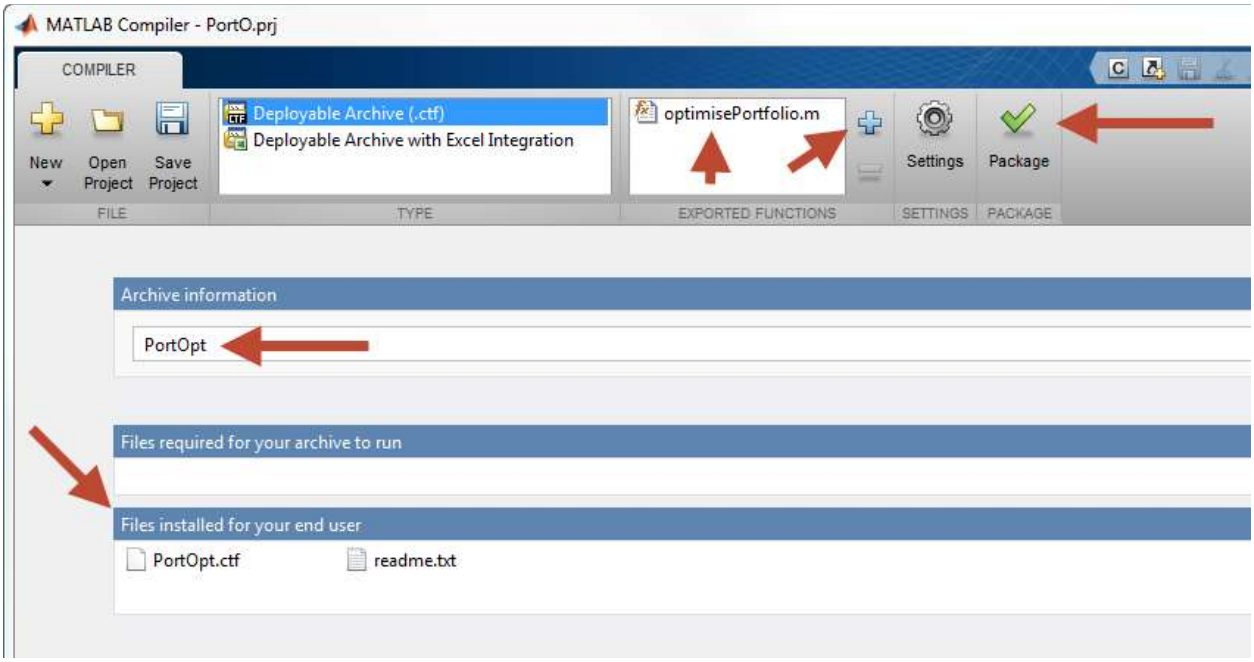
This represents a typical prototyping workflow in building data analytics algorithms in MATLAB.

When satisfied with the algorithm, the user can democratize the results of the analysis by taking the MATLAB code into production using TIBCO Spotfire.

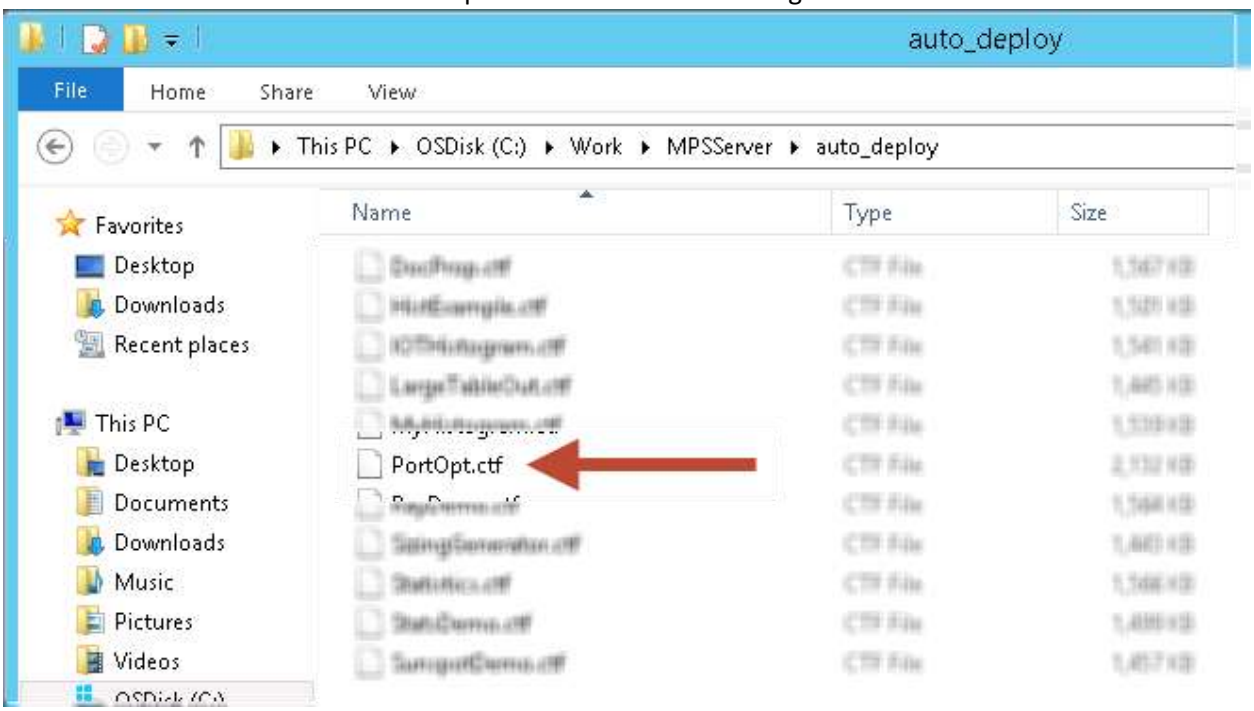
Production Workflow

The first step in taking the MATLAB function to production is to deploy the code using the MATLAB production server. This can be done using the deployment tool (*deploytool*).

10. Using the MATLAB Production Server Compiler, specify the function to be packaged and create the archive by selecting the “Package” button.



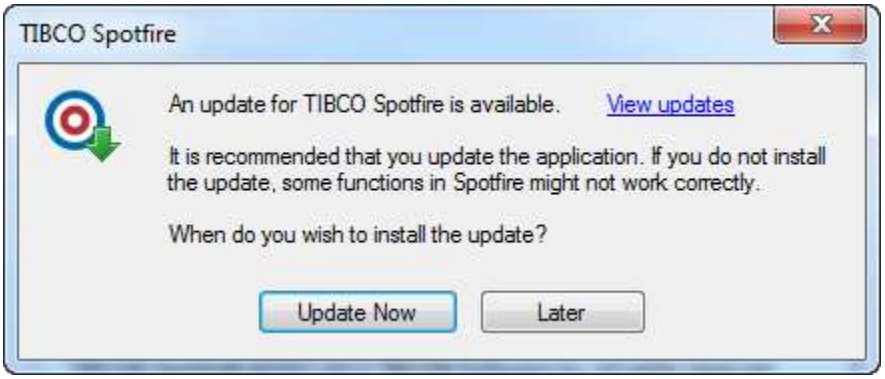
11. Once complete, copy the resulting '.ctf' file to the auto-deploy folder of MATLAB Production Server. Ensure that the MATLAB production Server is running.



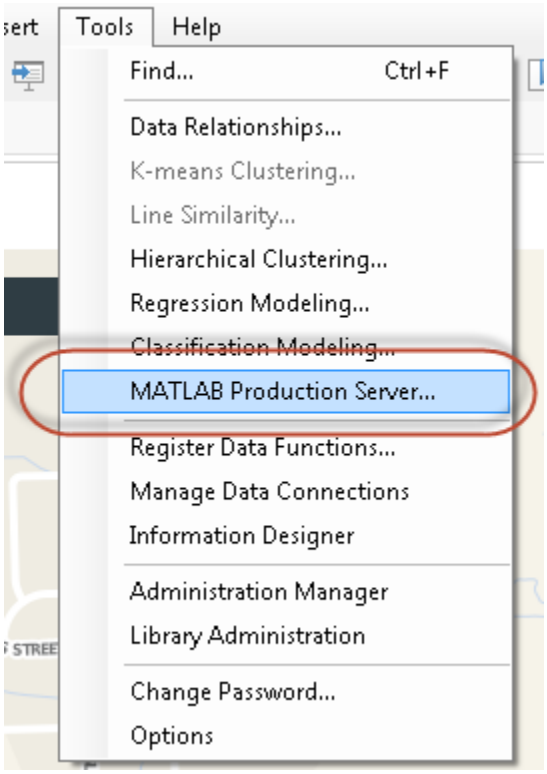
On the TIBCO Spotfire client machines

12. Restart any Spotfire clients and select "Update Now" to update the client with the MATLAB Production Server Interface for TIBCO Spotfire software.

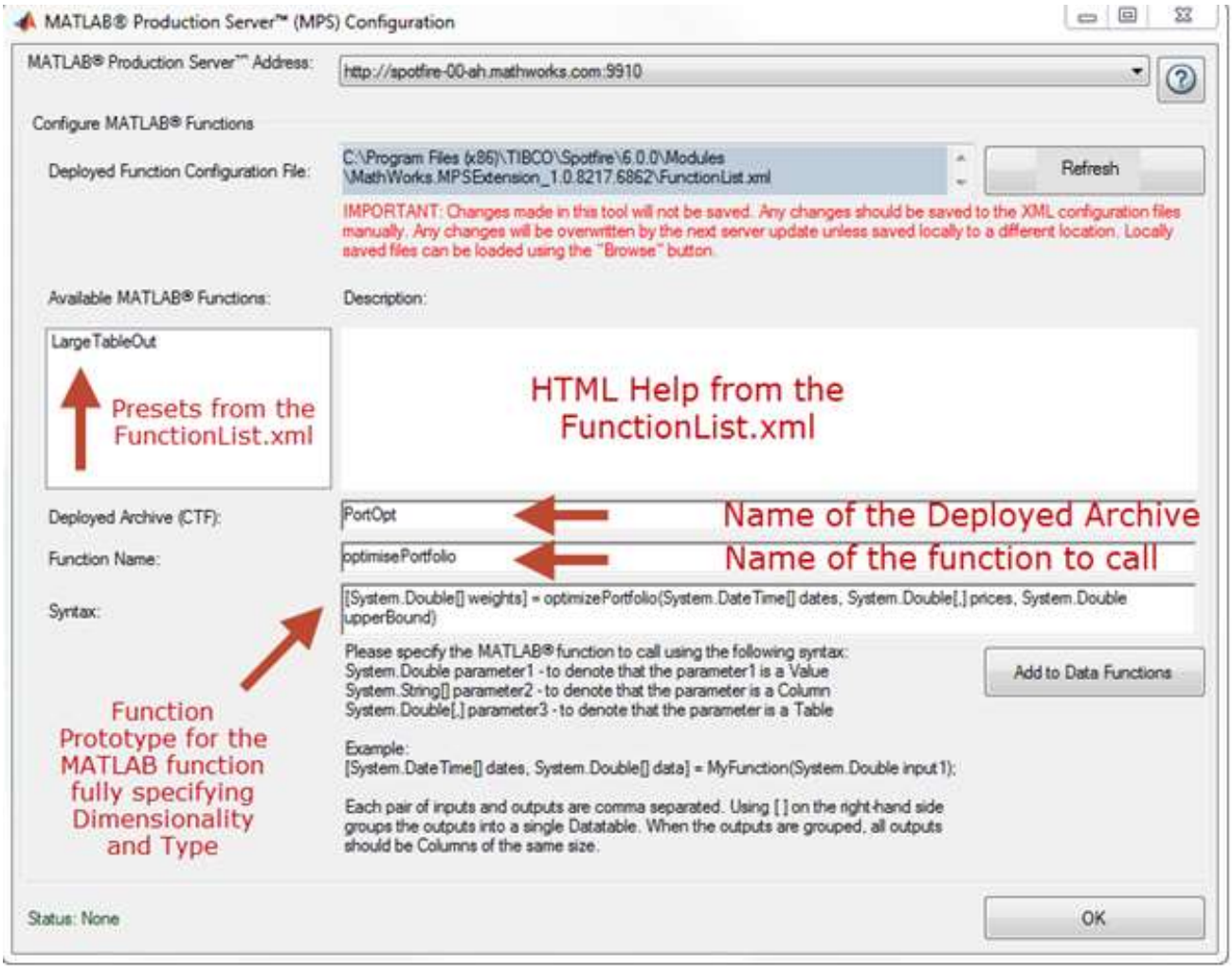
Installing the update will enable the Spotfire user to access the MATLAB Production Server tool.



13. From the Tools menu, select the "MATLAB Production Server" tool.



This brings up the tool that will allow configuration of the data function. The inputs include a selection of available MATLAB Production Server instances as configured earlier allowing users to partition their environment into development, test and production as per best practices.



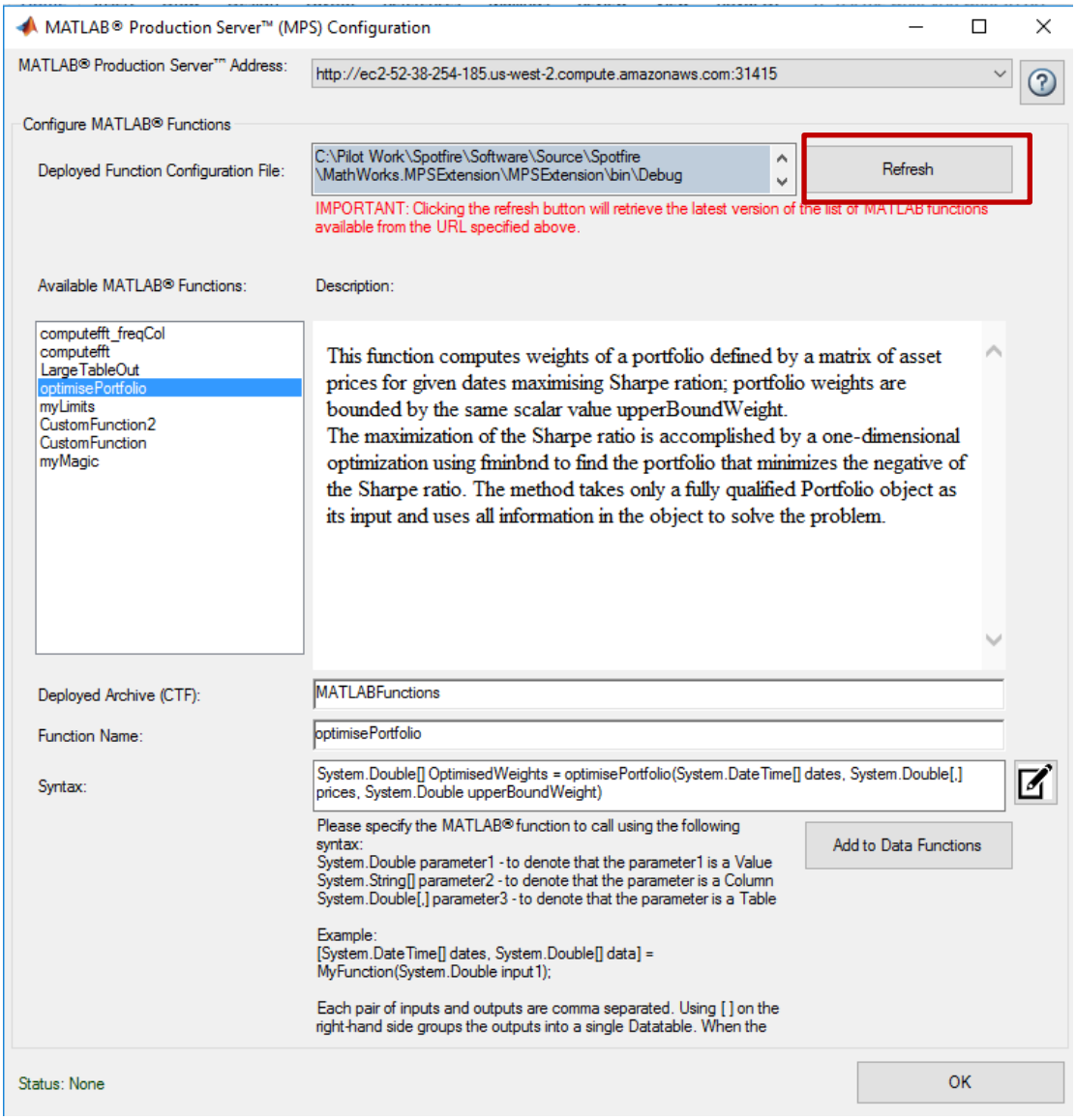
The figure above displays the UI for the MATLAB Production Server tool. The first field is the MATLAB Production Server Address. This is the URL of the machine where MATLAB Production server is hosted and is populated using information in the ServerConfiguration.xml file modified in step (3).

The second field is the Function Configuration file which contains the list of all MATLAB functions available. This is populated using information in the FunctionList.xml file modified in step (3).

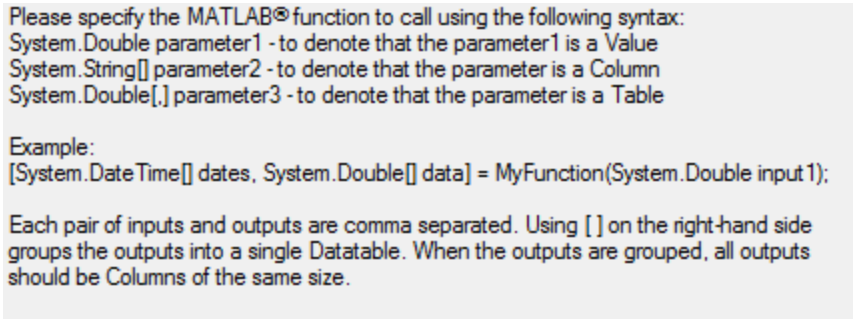
Modifying the configuration files after installation

Once the extension is installed, the two XML files(ServerConfiguration and FunctionList) are available locally on the end users machine in the path shown under 'Deployed Function Configuration File'(screenshot below). It is possible for the Spotfire end user to collaborate with the MATLAB developer and modify these files individually if changes are required. A second option is for the MATLAB developer to make the changes required and repackage the Spotfire updates and push it to all Spotfire end userd

The ‘Refresh’ button highlighted below allows the Spotfire end user to download the latest FunctionList.XML file directly from the MPS server. To enable this functionality, follow instructions in ***Appendix E***.



A prototype definition for the MATLAB function to be called enables marshaling of datatypes and dimensionalities between the two environments. The specification of the prototype uses the following syntax:



In our example, our MATLAB syntax:

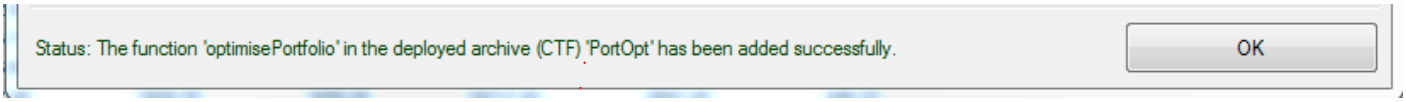
```
weights = optimisePortfolio(dates, prices, upperBoundWt)
```

would translate to:

```
[System.Double[] weights] = optimizePortfolio(System.DateTime[] dates, System.Double[,]
prices, System.Double upperBound)
```

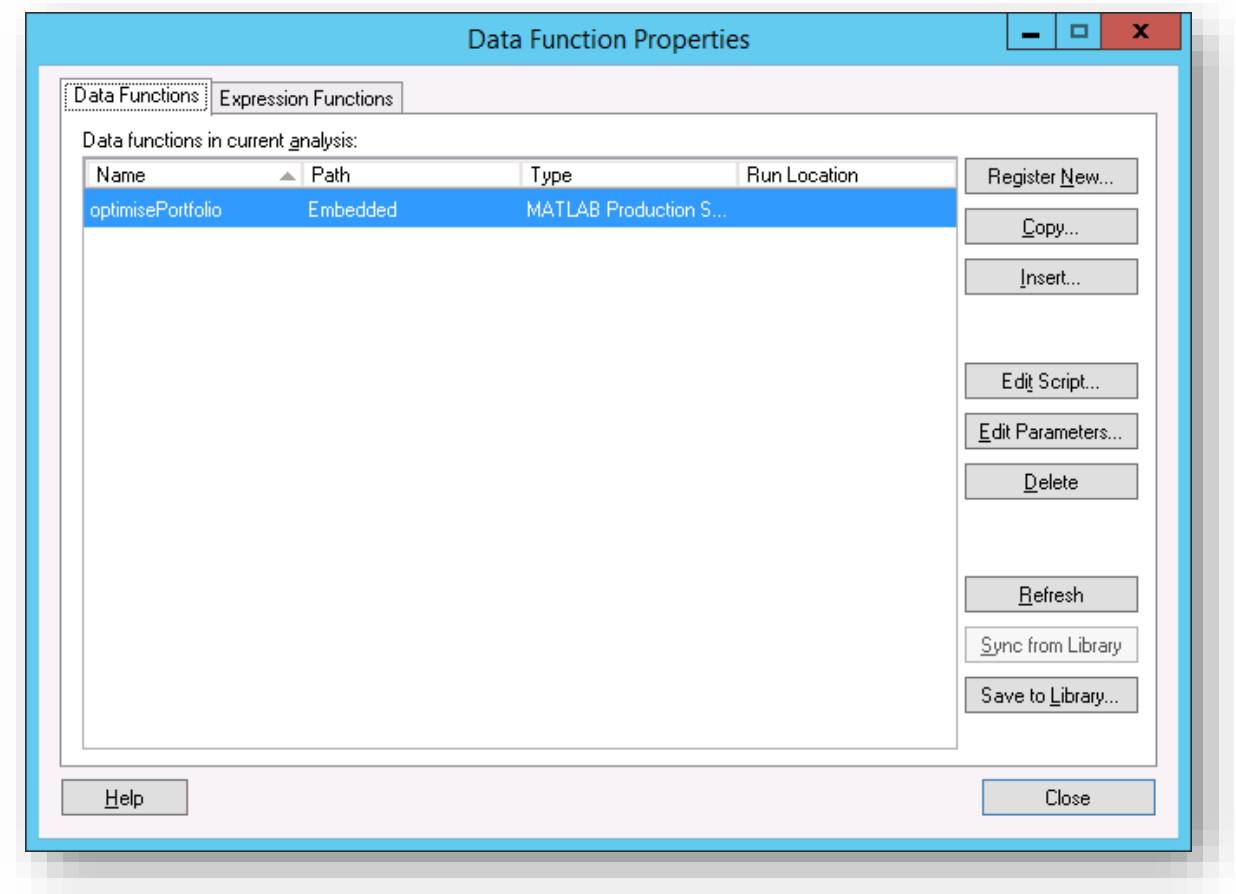
This can be roughly read as: *"Call the optimisePortfolio function providing a Column (vector) of date/time values, a Table (array) of Doubles, and a single scalar Double as an upper bound weight. The function will return a Column (vector) of weights as an output"*.

Pressing the *Add to Data Functions* button will add the algorithm to the current document's list of data functions.



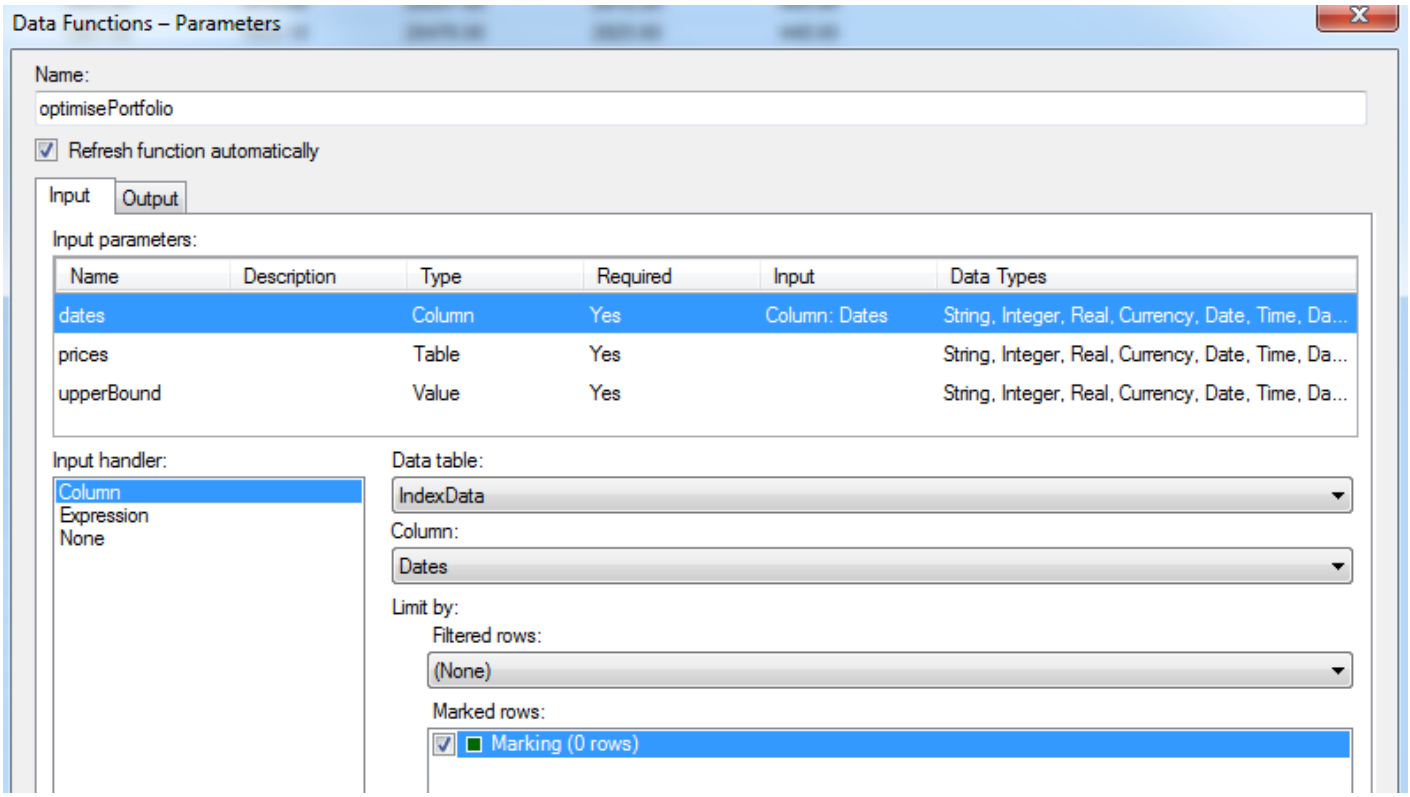
Clicking the OK button closes the GUI. The list of data functions is available in the documents properties.

14. From the “Edit” menu, select the “Data Function Properties”, and select “Edit Parameters”.

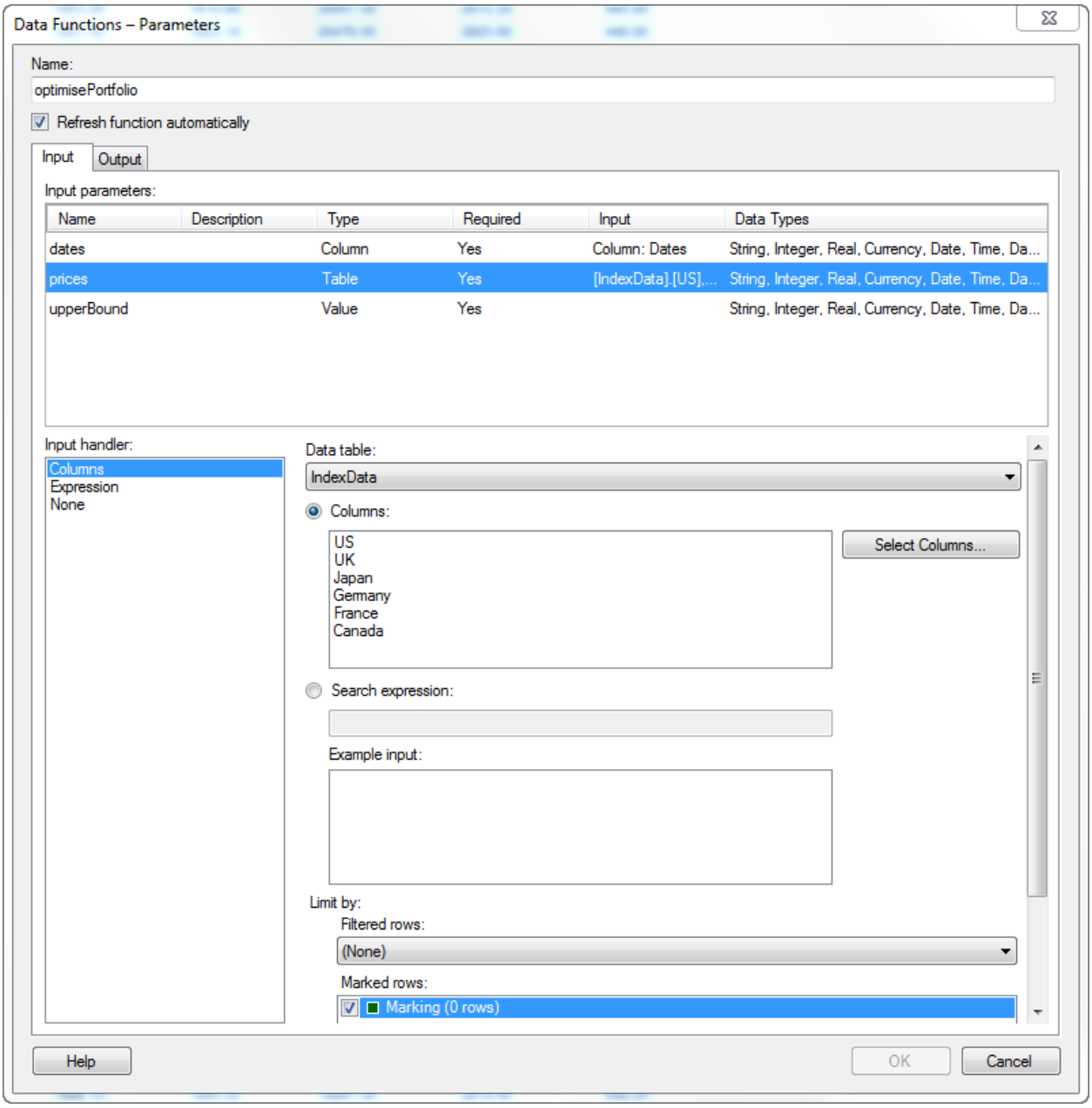


Map the data columns from the current document to the function’s parameters as appropriate (Input), and setup the data function to create a new Data Table using the results received from MATLAB Production Server (Output).

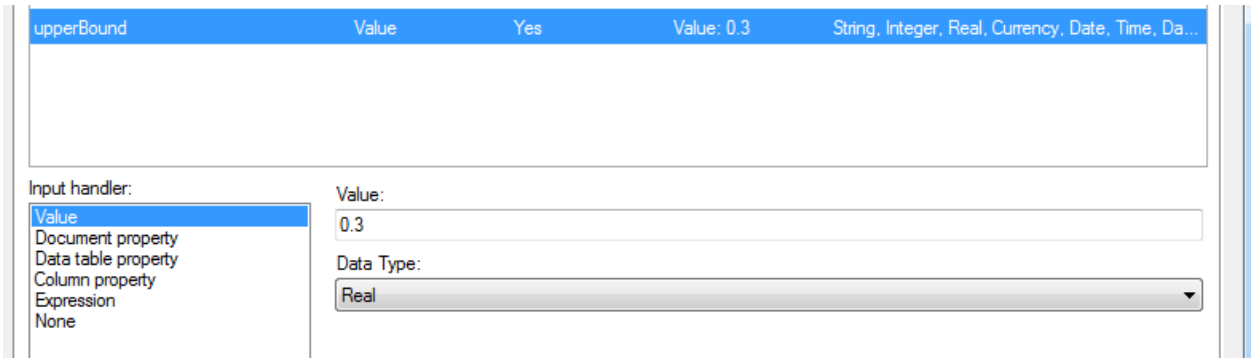
Each individual input and output can now be mapped, to the appropriate columns and table in the document. For example, the *dates* input is mapped to the *Dates* column of the *IndexData* table.



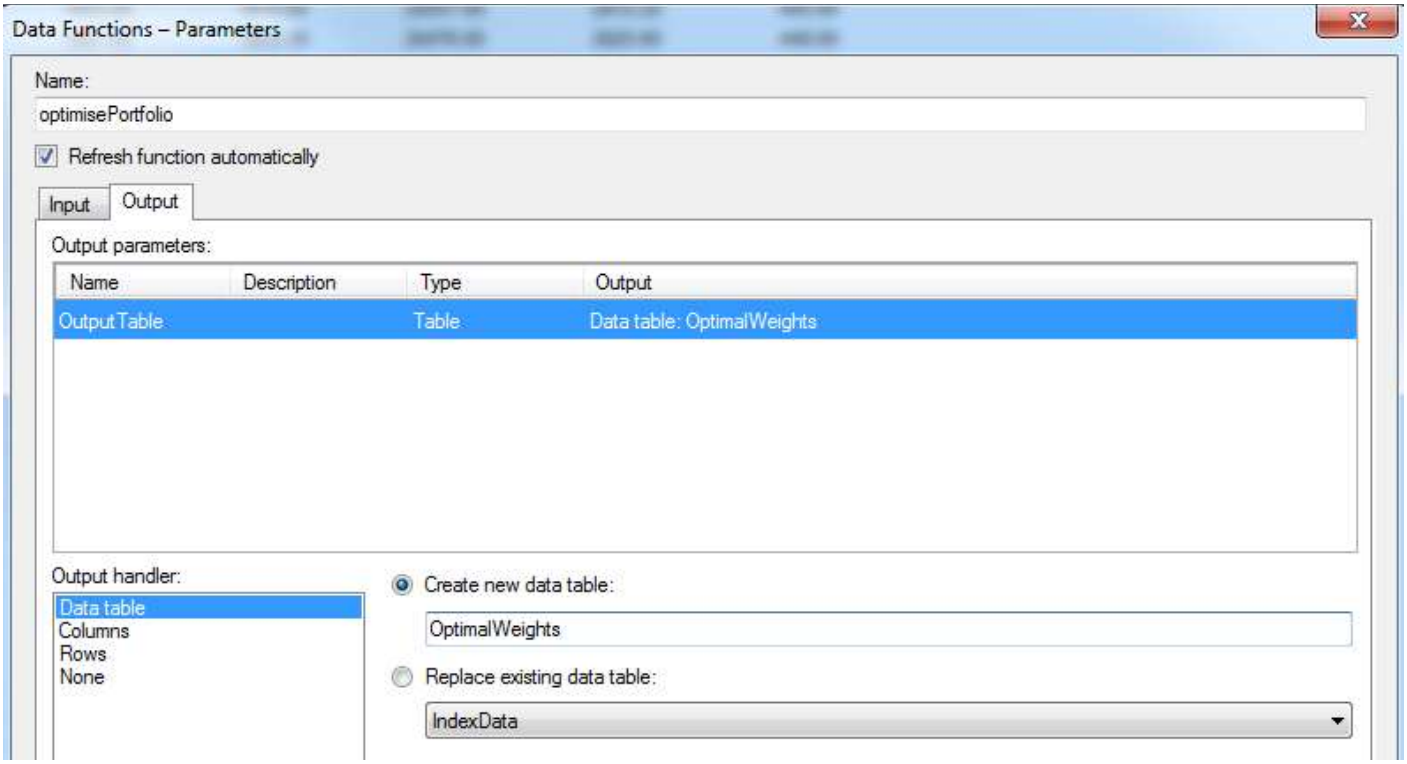
The *prices* input comes multiple columns from the *IndexData* table:



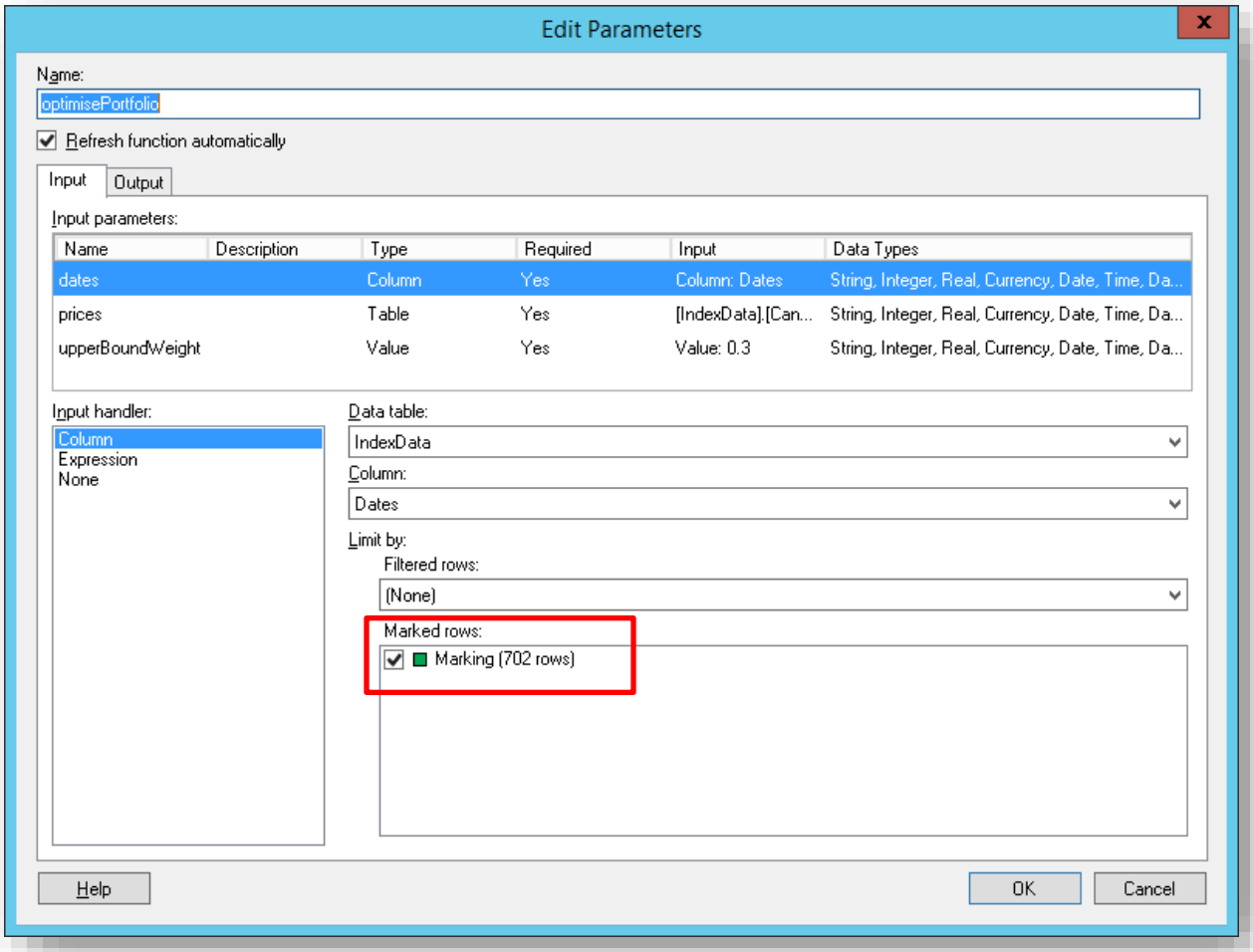
The *upperBound* input is mapped to a single scalar Double value 0.3.



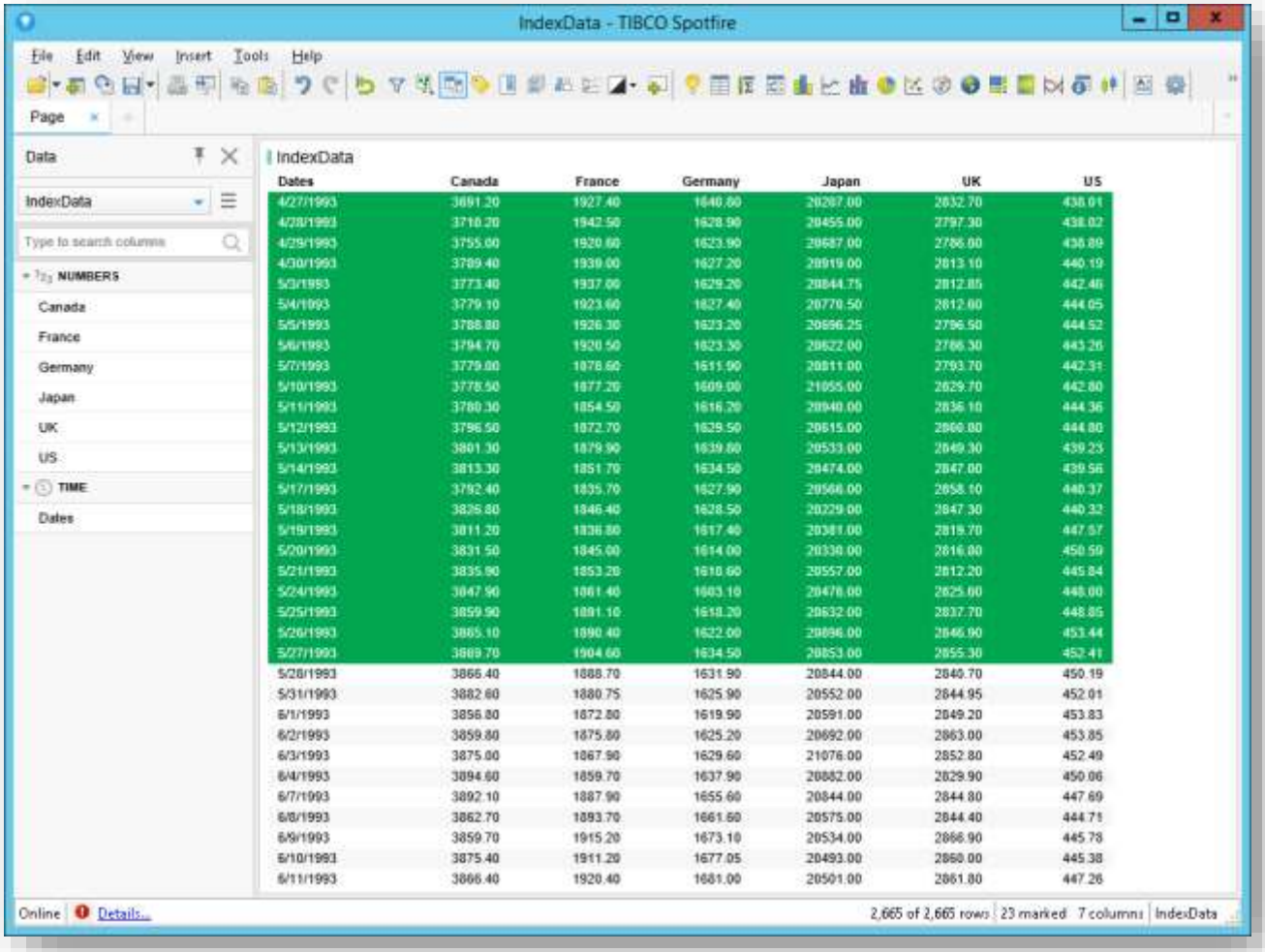
The output is configured to create a new data table.



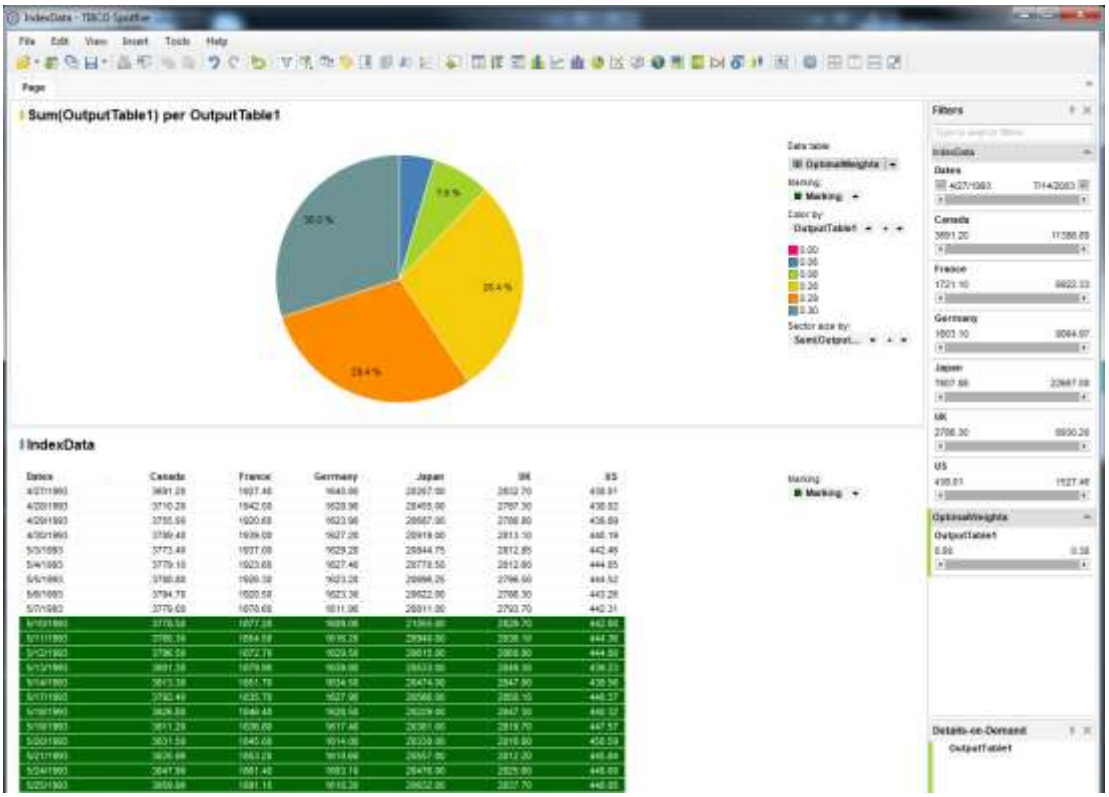
Finally, the entire function is marked to “Refresh automatically”. At this point, we have a fully configured function and any selection of the data will call the MATLAB function and create a new visualization. The MATLAB portfolio optimization code is now executed on every selected/marked point on the visualization in TIBCO Spotfire.


Edit Parameters
Name:
optimisePortfolio
Refresh function automatically
Input
Output
Input parameters:
Name
Description
Type
Required
Input
Data Types
dates
Column
Yes
Column: Dates
String, Integer, Real, Currency, Date, Time, Da...
prices
Table
Yes
[IndexData].[Can...
String, Integer, Real, Currency, Date, Time, Da...
upperBoundWeight
Value
Yes
Value: 0.3
String, Integer, Real, Currency, Date, Time, Da...
Input handler:
Column
Expression
None
Data table:
IndexData
Column:
Dates
Limit by:
Filtered rows:
(None)
Marked rows:
Marking (702 rows)
Help
OK
Cancel

15. After “marking” or selecting a section of data, the new Data Table reference will be available within the document for use with any of the Spotfire visualization tools.



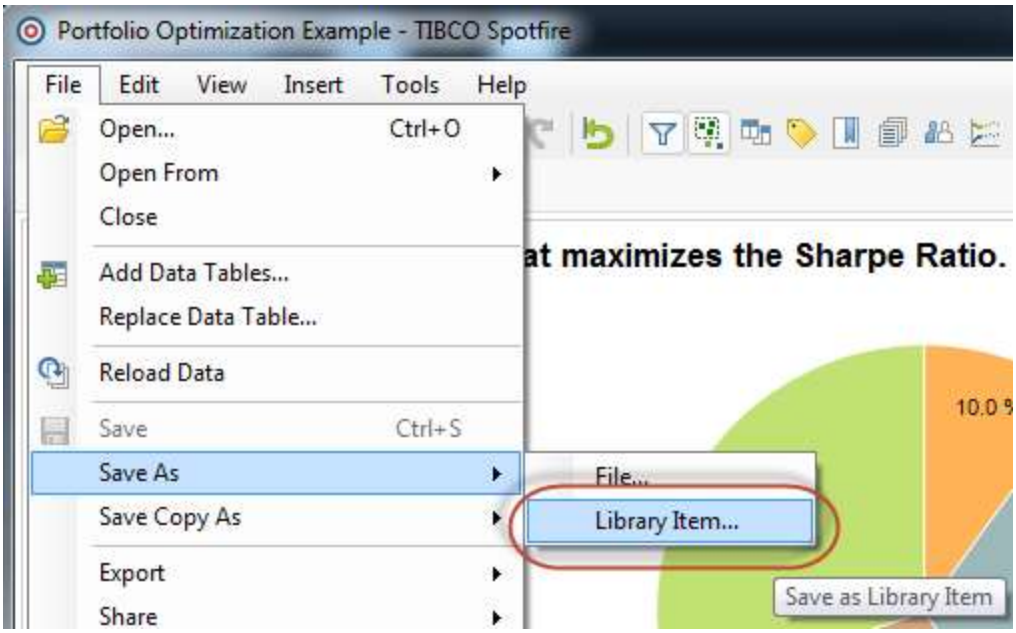
16. Add new visualizations using the “Output” data



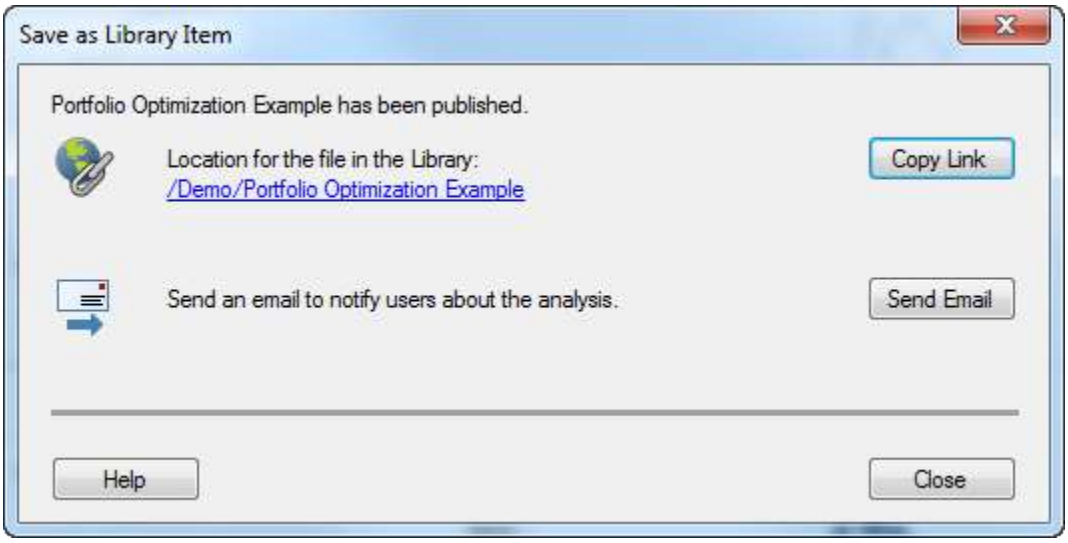
17. Finish your document and publish it for use by others using the web and mobile clients.

Web Users

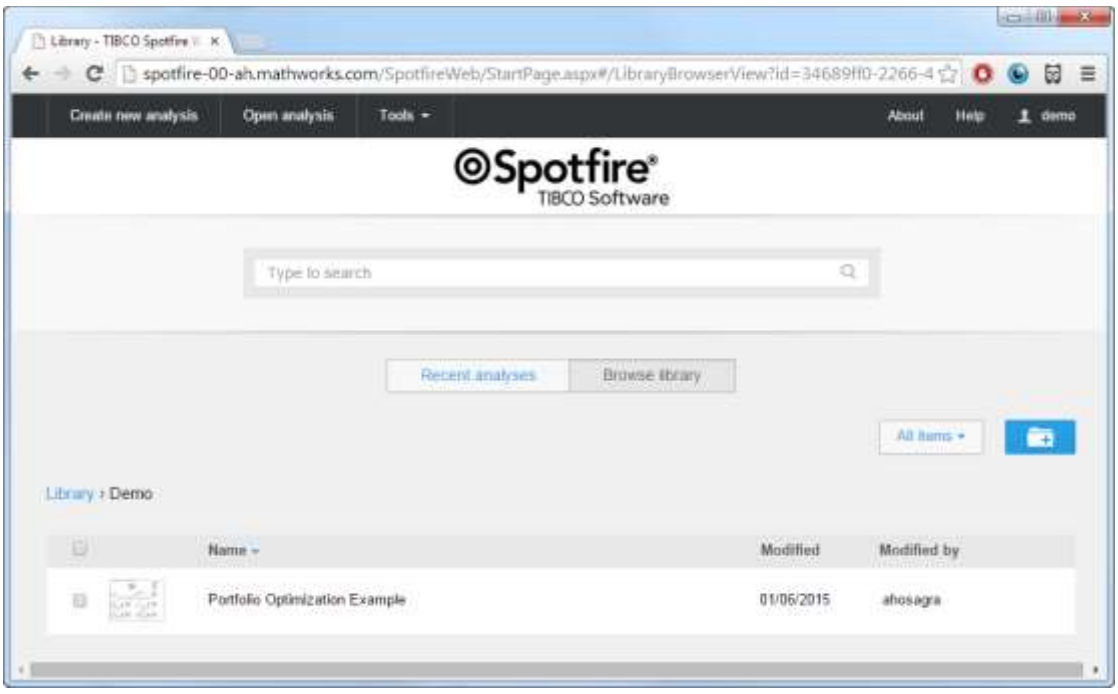
Making the analysis developed above available on the network via a Browser is now possible using the TIBCO Spotfire Web Player. The Web player uses the same extension code to call MATLAB to provide interactive dashboards available off a web browser. To make the analysis available via to Web users who will not need any special client side tooling. To do this, the analysis is saved to the Library making it available to all users of the Spotfire Server.



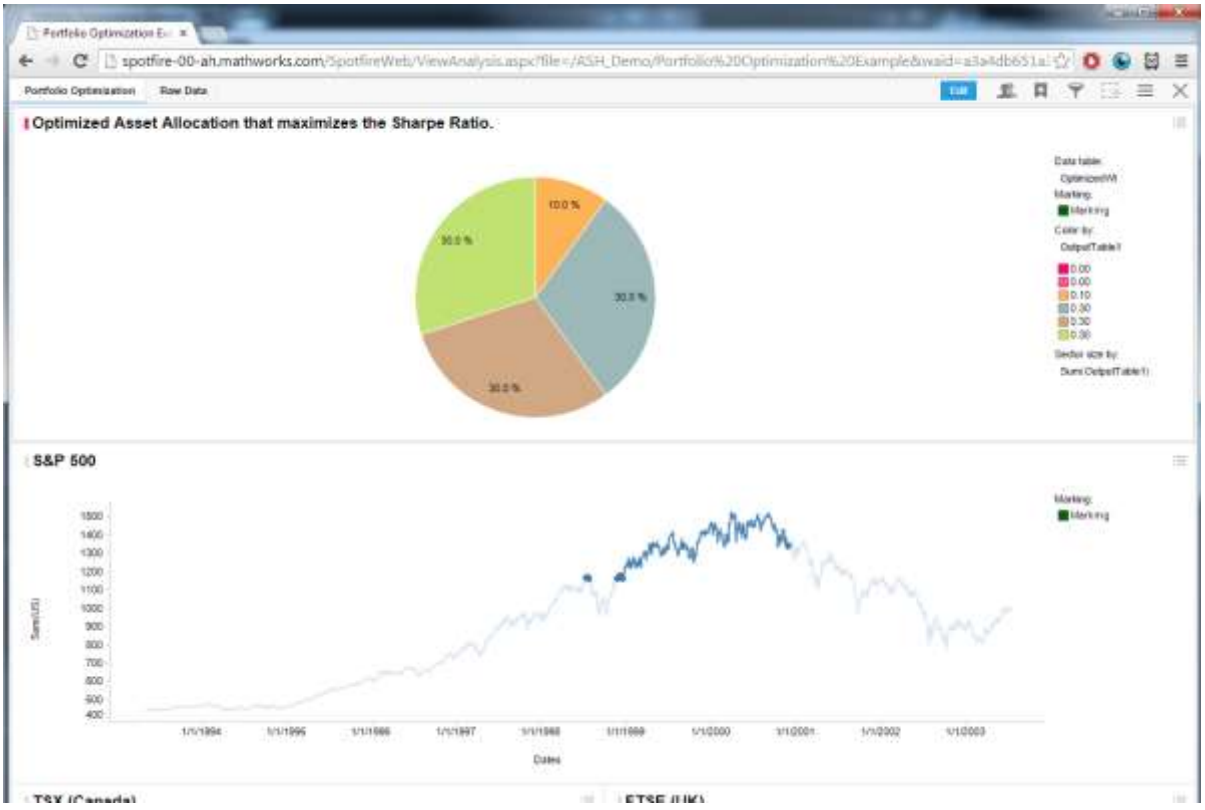
The tool will prompt for a filename and a few other minor details such as description. The publication process should be close to a few clicks at most.



The Spotfire WebPlayer now provides the analysis on a regular JS enabled internet browser. Shown below is MATLAB powered interactive analysis available in a standard Web Browser (Chrome).



Clicking on the analysis brings up the MATLAB powered dashboard in a Web Browser.



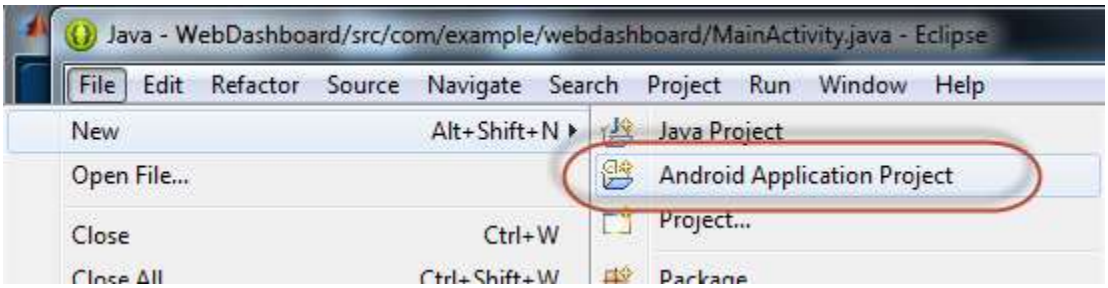
This technique enables multiple users to concurrently access the analysis. Every click of the interactive visualization calls our optimization MATLAB code. The scalability of the back-end infrastructure is possible by using load-balancers both for the TIBCO Spotfire servers as well as the MATLAB Production Servers as per recommended best-practices for the IT setup of these infrastructure stacks.

Mobile Platform Users

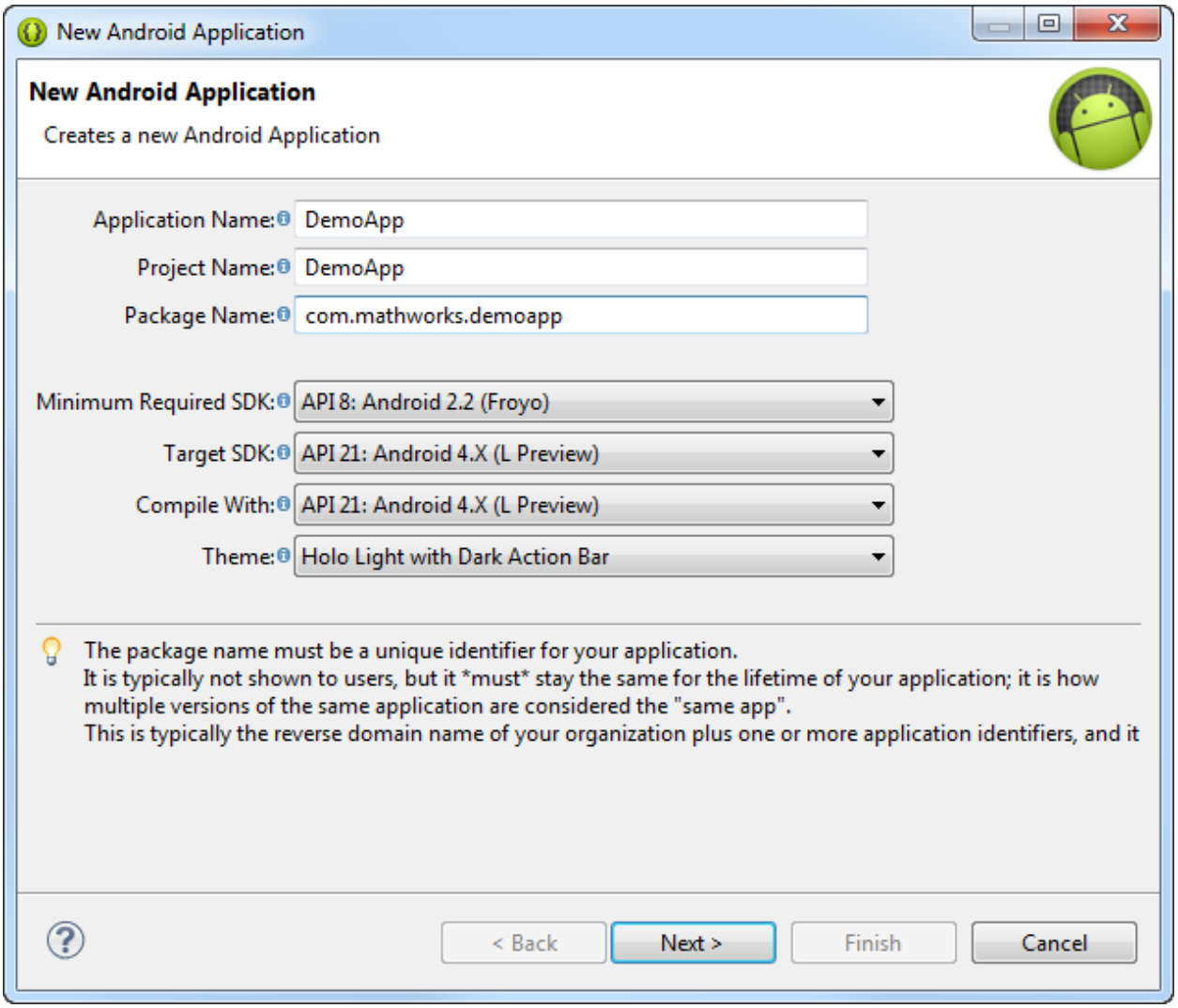
The TIBCO Spotfire stack offers tools such as tibbr® [7] for the collaboration aspects of making the results of data analysis widely available. However, given the availability of the MATLAB powered dashboard as a pure-client side visualization, it is also possible to create custom branded applications that leverage the availability of the analysis as a Web applications that leverage the WebView in iOS [8] and Android [9] powered devices.

As an example, we will create a custom Android application that exposes the MATLAB computation through an Android device. This example uses the Android Software Development Kit [10].

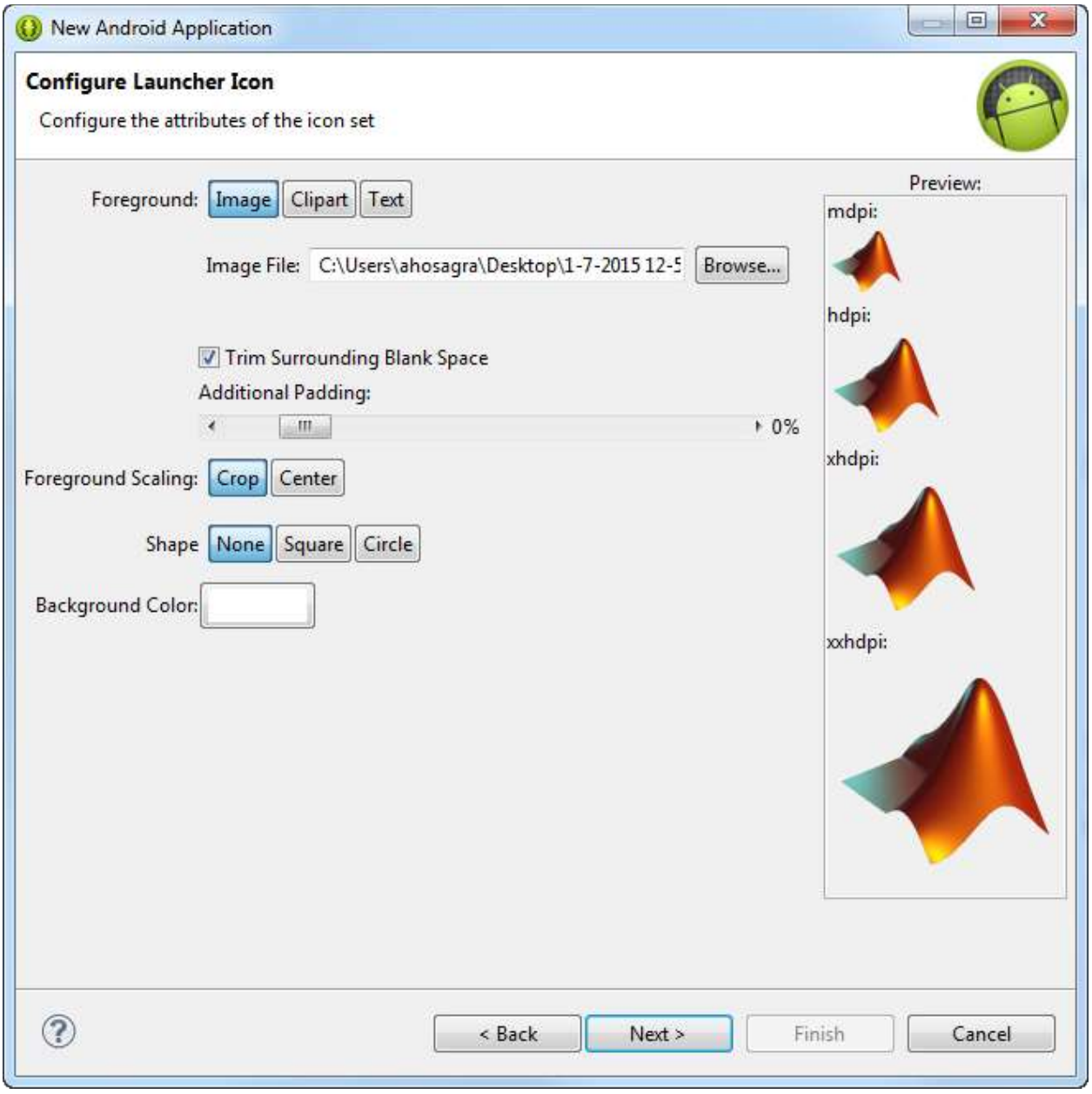
Start a new Android application project from the menu.



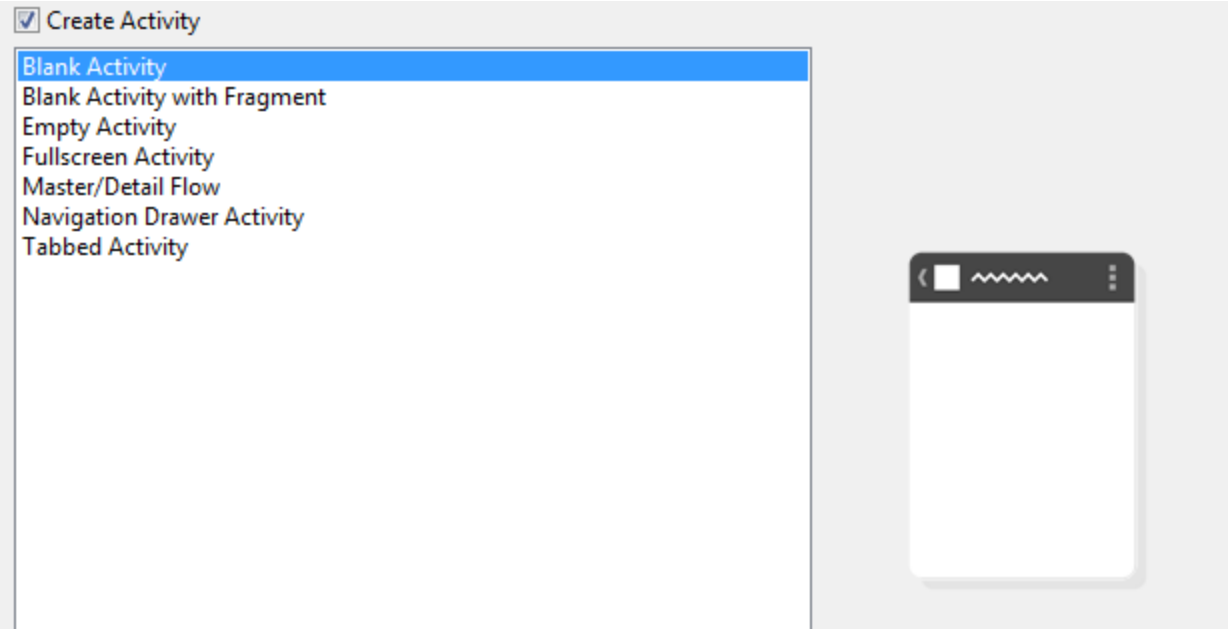
Provide a name for the application and package to configure the application.



It is also possible to brand the application with custom look/feel themes but discussion of that is outside the scope of this document.



For this example, we will just use a blank template:



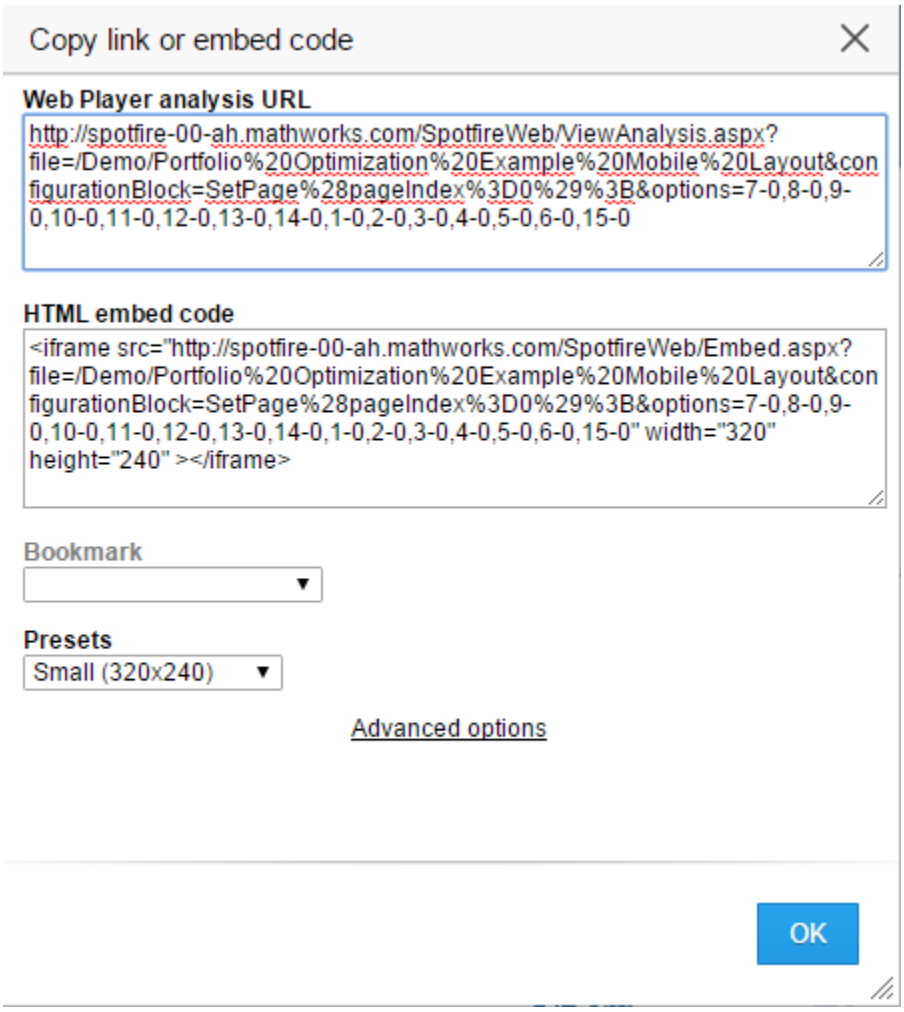
The analysis will be made available in a WebView within the application. Adding this to the main layout file - *activity_main.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

For this example, we will bring up a WebView on creation of the application. In the *MainActivity.java* WebView can be created and rendered.

```
/* Import the WebView from the Android WebKit */
import android.webkit.WebView;
import android.webkit.WebSettings;

/* Create and load our analysis */
  WebView myWebView = (WebView) findViewById(R.id.webview);
```

Pointing the WebView to use our analysis as published to the TIBCO Web Player. The URL in this case can be obtained from the Web interface.




```
        myWebView.loadUrl("http://spotfire-00-
ah.mathworks.com/SpotfireWeb/ViewAnalysis.aspx?file=/Demo/Portfolio%20Optimization%20
Example%20Mobile%20Layout&configurationBlock=SetPage%28pageIndex%3D0%29%3B&options=7-
0,8-0,9-0,10-0,11-0,12-0,13-0,14-0,1-0,2-0,3-0,4-0,5-0,6-0,15-0");
```

Enabling Javascript allows interactivity of the rendered analysis.

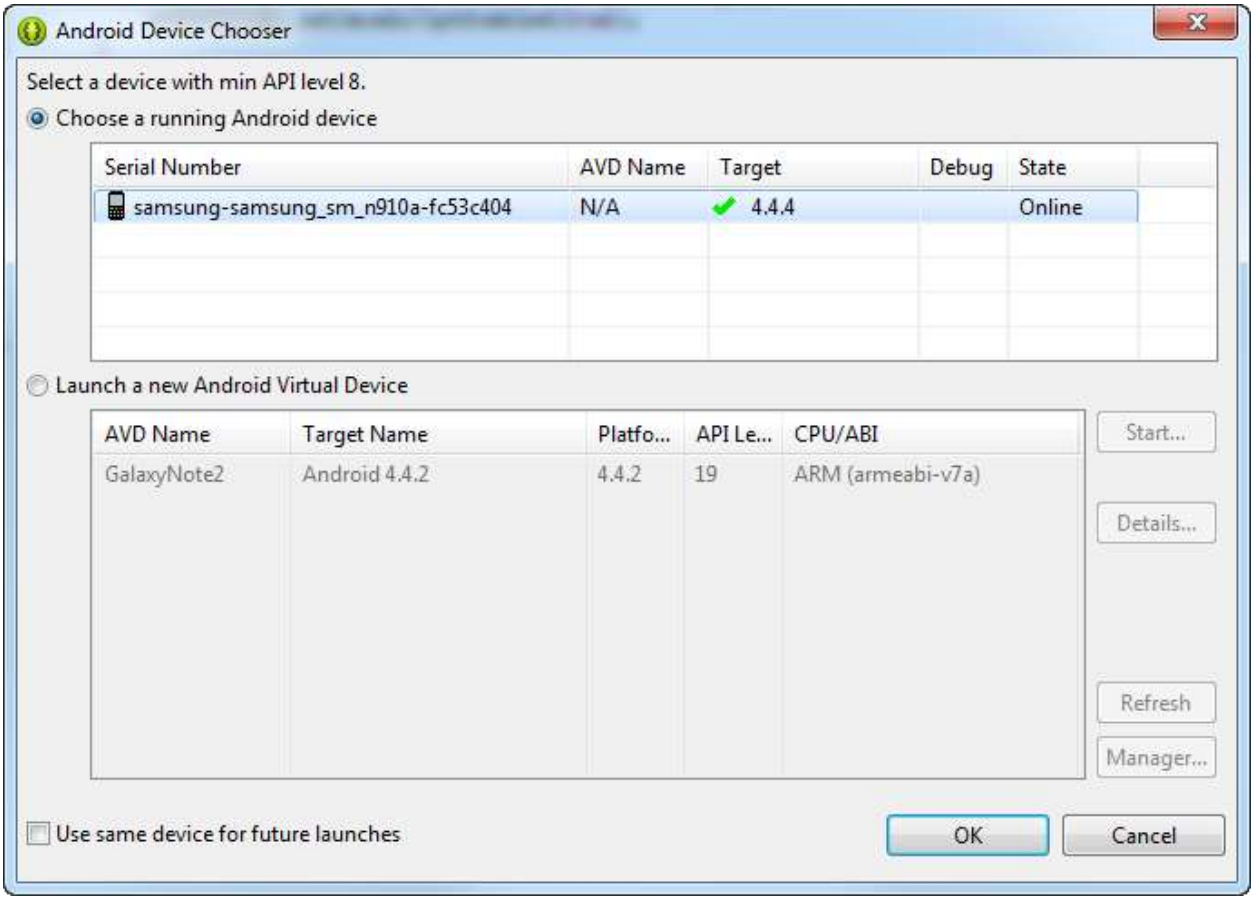
```
      /* Enable Javascript support */
        WebSettings webSettings = myWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
```

Finally, the application is configured to request and allow internet access.

```
      <manifest ... >
          <uses-permission android:name="android.permission.INTERNET" />
          ...
      </manifest>
```

Our application is ready to use. In reality, the layout of a real application will be more complete and while it is possible to bind javascript code to the Android code, this topics are beyond the scope of this

document. On building and running the application, we have our view rendered to the Android device. For this example, we will use a mobile phone.



The custom branded application is now available as an Android application.

Every click and interaction on the mobile device calls the MATLAB code to perform a portfolio optimization of the data over the selected date range. In a real application, it would be necessary to optimize the view for a mobile platform by adjusting the layout of the analysis.

A similar demo would be possible for an iOS device, a detailed discussion of which is outside the scope of this document.

In conclusion, this architecture enables MATLAB powered analytics to be accessed in a reliable and scalable manner across a wide variety of platforms ranging from desktop clients, servers, web browsers and even mobile devices.

References

[1] TIBCO Spotfire® - Business Intelligence Analytics Software & Data Visualization (2014, December 23rd). Retrieved from http://spotfire.tibco.com/

[2] MATLAB – The language of technical computing (2014, December 23rd). Retrieved from http://www.mathworks.com/products/matlab/

[3] MATLAB Production Server (2014, December 23rd). Retrieved from http://www.mathworks.com/products/matlab-production-server/

[4] TIBCO Spotfire® Data Sources (2014, December 23rd). Retrieved from http://spotfire.tibco.com/resources/spotfire-data-sources

[5] Portfolio Optimization (2014, December 23rd). Retrieved from http://en.wikipedia.org/wiki/Portfolio_optimization

[6] Sharpe Ratio (2014, December 23rd). Retrieved from http://en.wikipedia.org/wiki/Sharpe_ratio

[7] tibbr® - The social network for work (2014, January 7th). Retrieved from http://www.tibbr.com/

[8] Getting started with iOS Web Apps (2015, January 7th). Retrieved from https://developer.apple.com/library/safari/referencelibrary/GettingStarted/GS_iPhoneWebApp/_index.html

[9] Building Web Apps in WebView (2015, January 7th). Retrieved from http://developer.android.com/guide/webapps/webview.html

[10] Android SDK (2014, December 23rd). Retrieved from http://developer.android.com/sdk/index.html

[11] Spotfire Server Environment – Clustering and Failover (2015, January 7th). Retrieved from: http://stn.spotfire.com/stn/Platform/SpotfireServer.aspx

[12] MATLAB Production Server – Performance Optimization and Scalability (2015, January 7th). Retrieved from: http://www.mathworks.com/products/matlab-production-server/features.html#performance-optimization-and-scalability

Contact Information

Dave Oswill (508-647-3011)
Product Marketing Manager
Dave.Oswill@mathworks.com

Arvind Hosagrahara (310-819-3960)
Principal Technical Consultant
Arvind.Hosagrahara@mathworks.com

Appendix A: Tool installation

Each of the tools in the stack that comprise of the reference architecture come with its own system requirements and installation instructions. The coverage of all the instructions is outside the scope of this documents but below is a link to installation guides of each of the core components.

- MATLAB
  (http://www.mathworks.com/help/install/index.html)
- MATLAB Production Server
  (http://de.mathworks.com/help/pdf_doc/mps/mps_install.pdf)
- TIBCO Spotfire Server
  (https://docs.tibco.com/pub/spotfire_server/6.5.0/TIB_sfire_server_doc_ServerInstallationManual.pdf)
- Installing TIBCO Web Player
  (https://docs.tibco.com/pub/spotfire_web_player/6.5.2/doc/pdf/TIB_sfire_webp_6.5.2_InstallationManual.pdf)

An abbreviated set of notes in installing the tools to support the demonstration workflow in this document can be found at *./Documentation/Installation_Instructions.docx.*

Appendix B: Rebuilding the Spotfire Extension

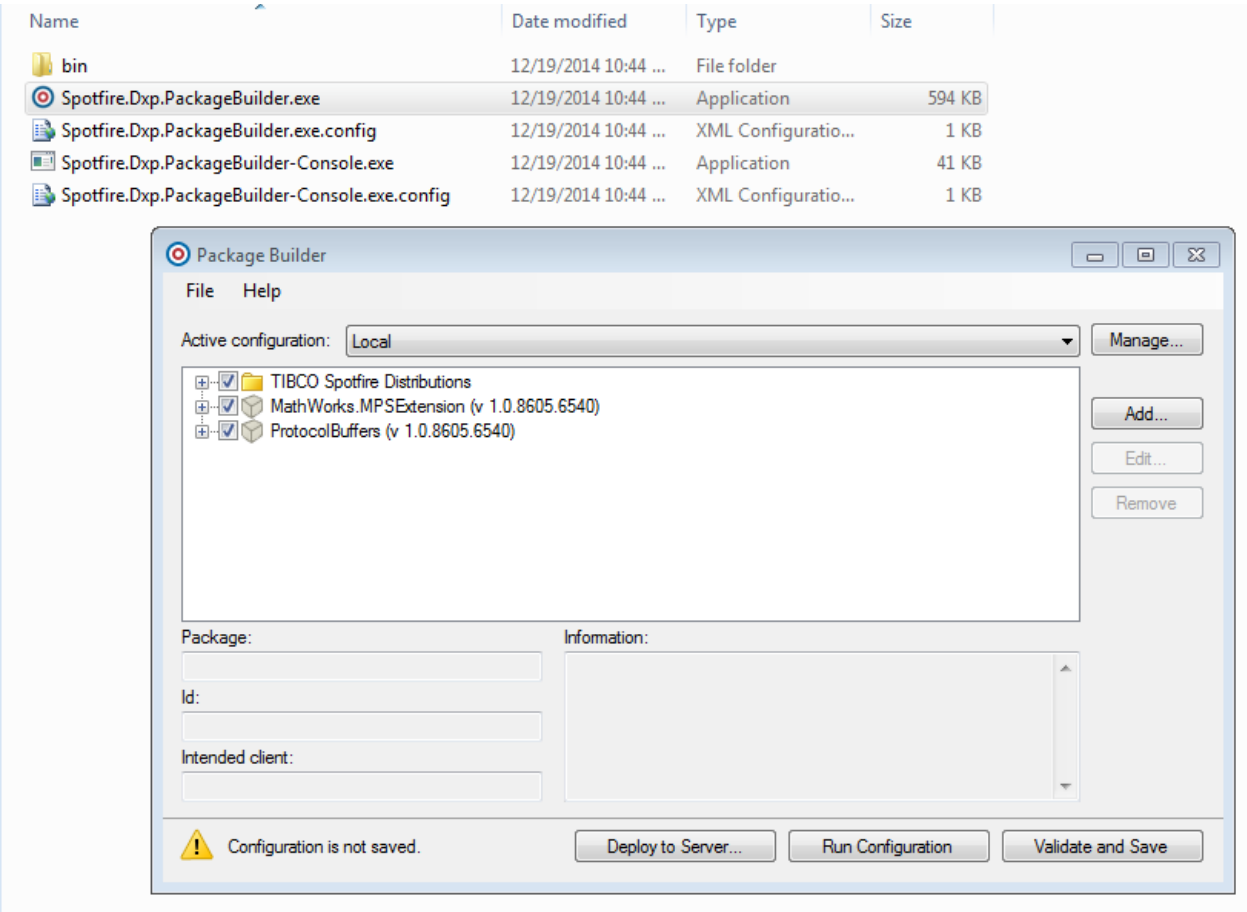
The *MathWorks.MPSExtension* was built using the .NET framework 4.5 and Visual Studio 2012.

The entire source code for the extension can be found in the *./Software/Source/Spotfire/MathWorks.MPSExtension* folder. The .sln file in the folder opens the extension and users can modify and build the extension using the *Build->Rebuild Solution (F6)* menu item.

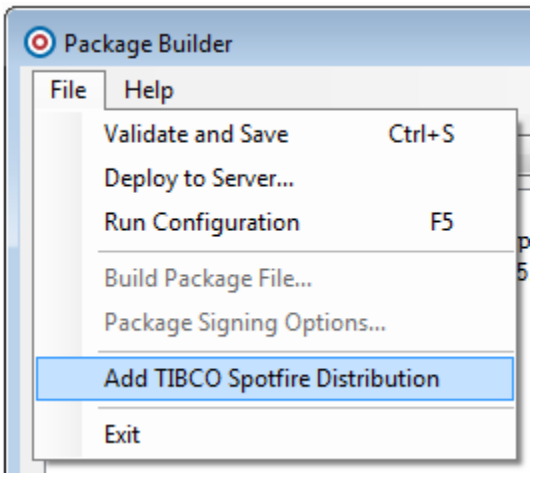
Depending on the installation of the MATLAB Production Server product the client DLL may need to be resolved in the references. This can be done by removing the existing DLL reference and re-adding the DLL that is packaged with the installation of the MATLAB Production Server.

When rebuilt, depending on the configuration, the RELEASE or DEBUG folder contains the binaries that are used to package and create the .SPK file used for distribution. This allows the user to configure the FunctionList.xml or the ServerConfiguration.xml to point to their MPS installation.

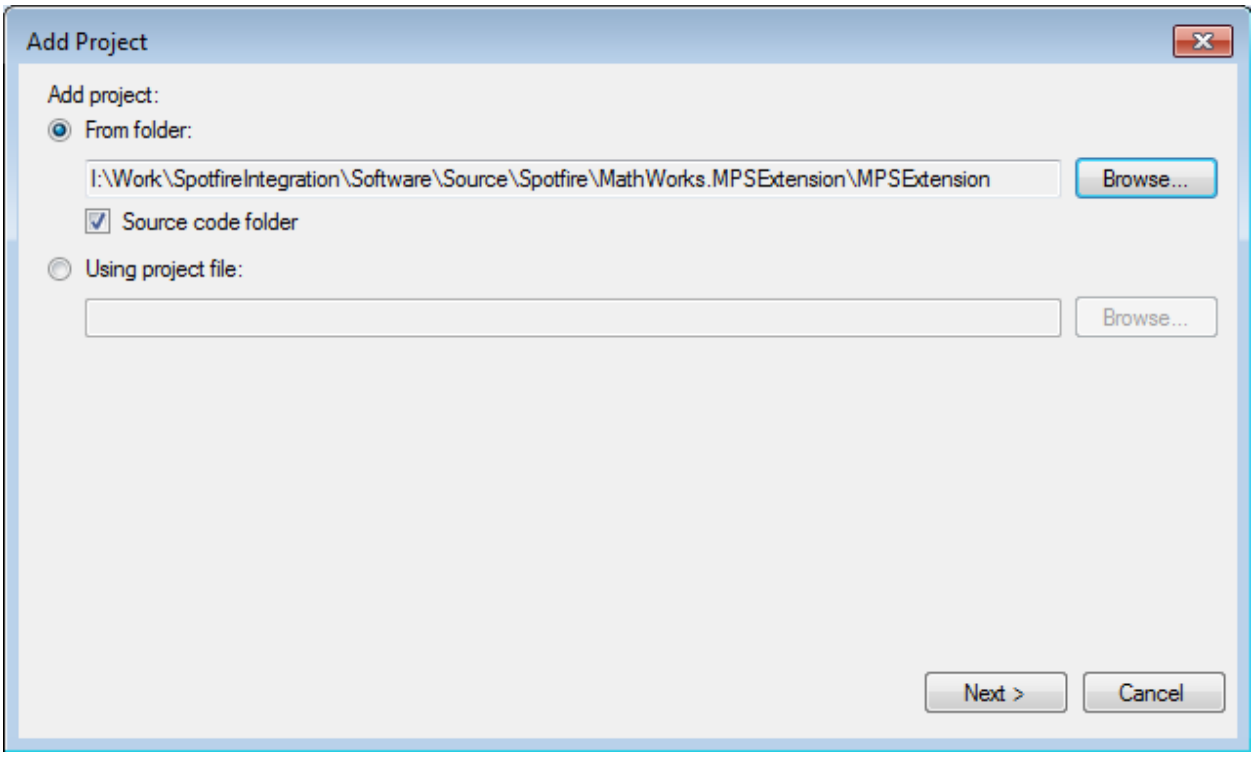
To build a new deployable Spotfire Package, start the Package Builder tool that bundles with the Spotfire SDK tools.



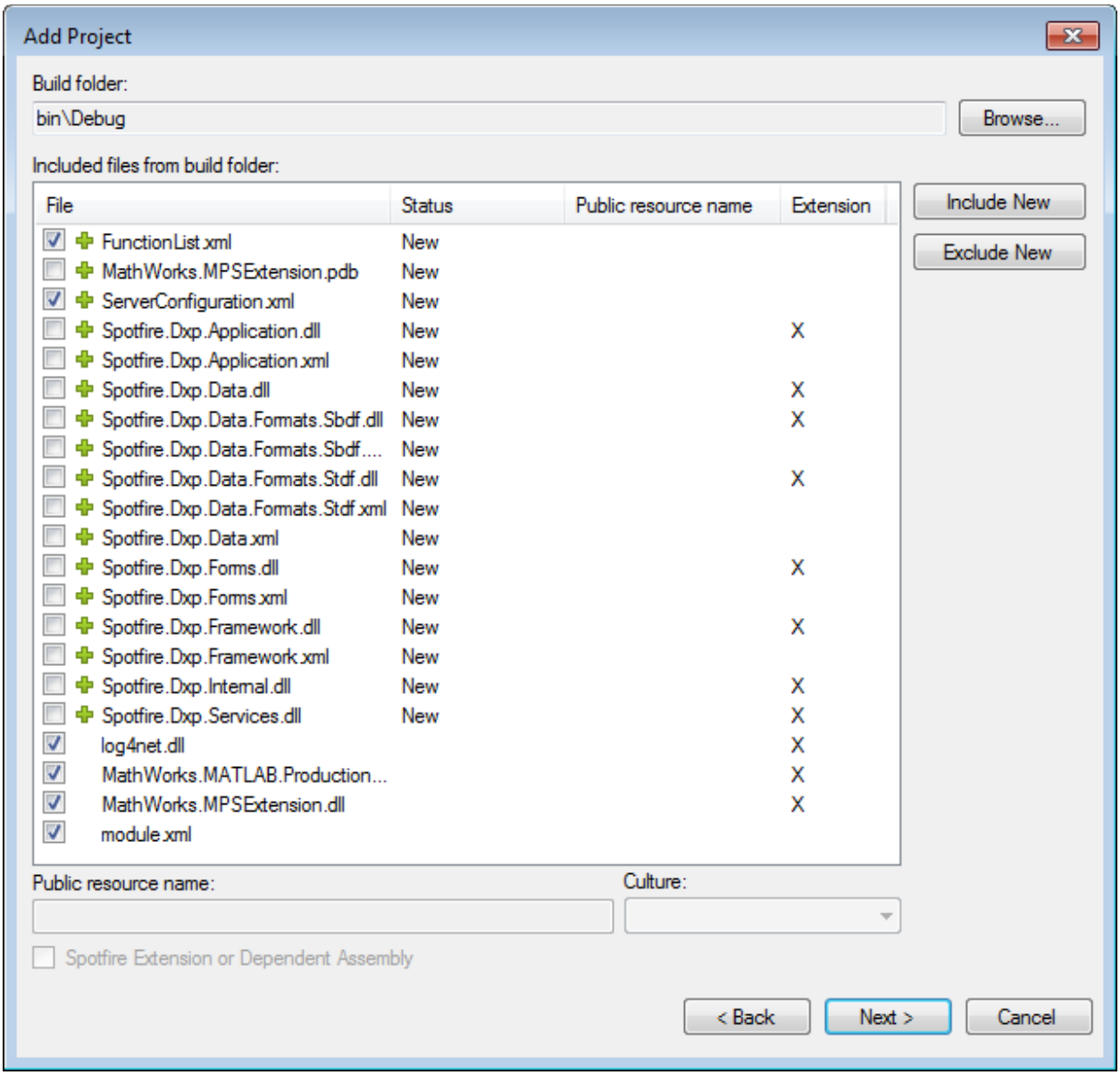
First, add a Spotfire distribution by clicking on the File > Add Spotfire Distribution menu item:



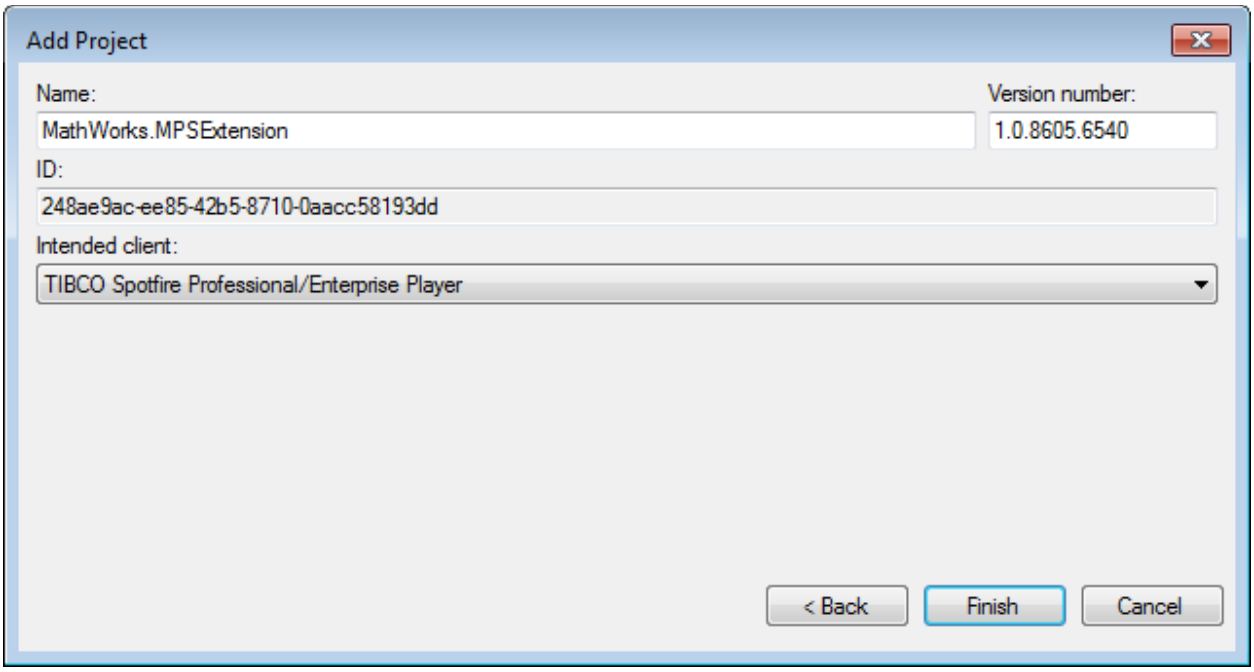
To rebuild the interface, the rebuilt binaries are configured in the tool using the Add button to a new project.



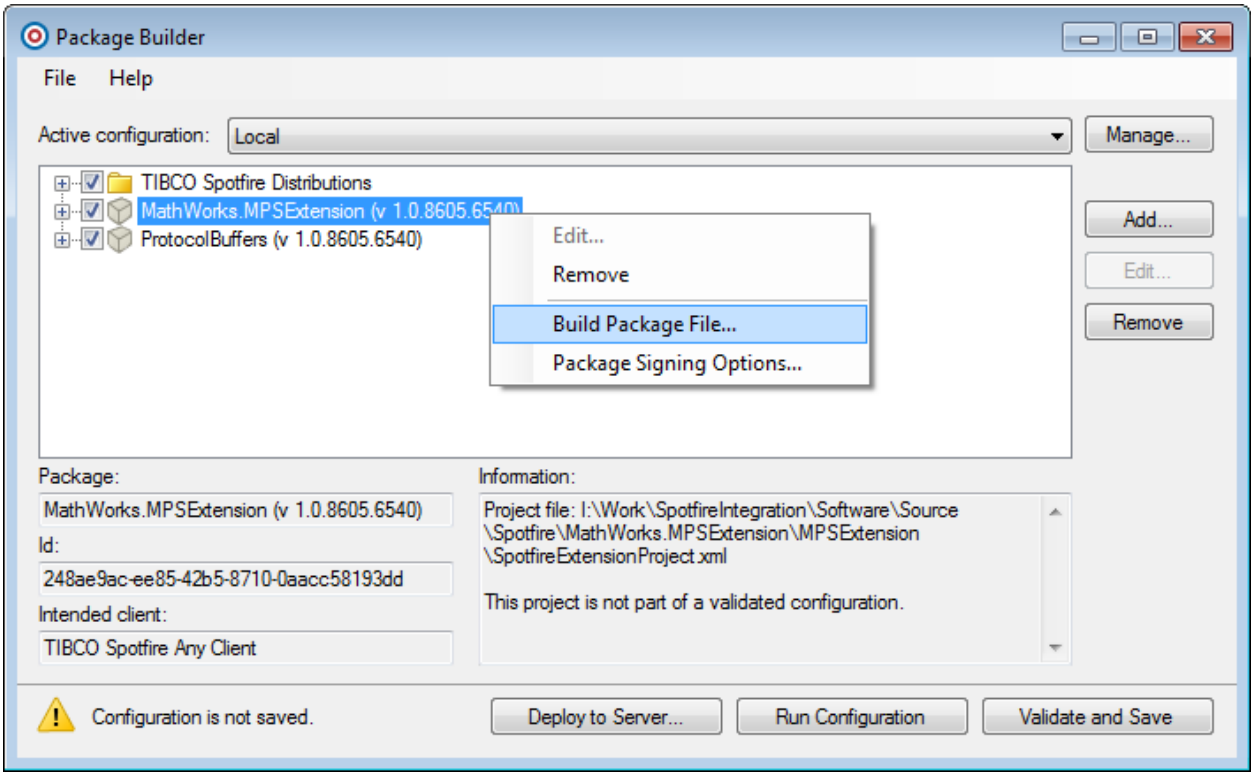
Follow the wizard by clicking the Next button and creating a new module.xml file (or choosing an existing file if you are repeating the process). Finally select the sources to bundle.



Adjust the desired version number and finish the process.



Right clicking on the project to build a new package file (.spk) for distribution with the Spotfire Server.

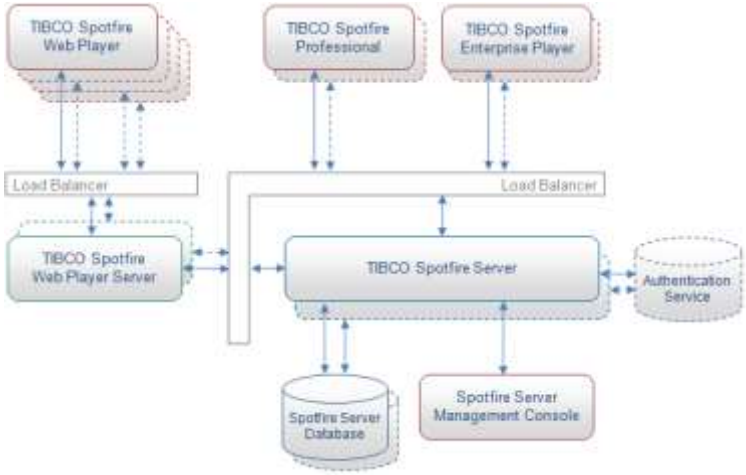


The resulting SPK file can be saved and deployed to the server. Alternatively, the 'Deploy to Server' button could be used to directly deploy the package to the Spotfire Server and make it available to all clients.

Appendix C: Performance Tips

Both the MATLAB Production Server and TIBCO Spotfire product can be installed in clustered configuration for higher performance, availability. The discussion of these configurations for a high-reliability business-critical application that conforms to best practices for performance, fail-over, etc. is outside the scope of this document. In a nutshell, the two parts of the stack can be built up using their own clustered configuration.

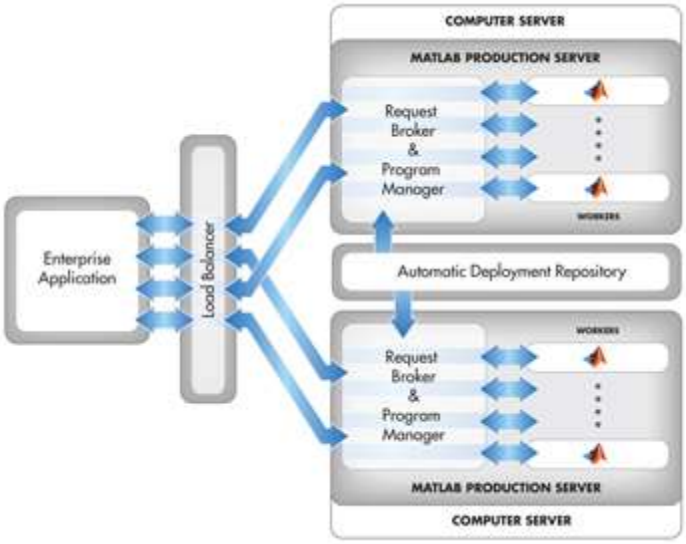
TIBCO Spotfire in a clustered configuration



Please see the clustering and fail-over configuration in the Spotfire Documentation [11]: http://stn.spotfire.com/stn/Platform/SpotfireServer.aspx

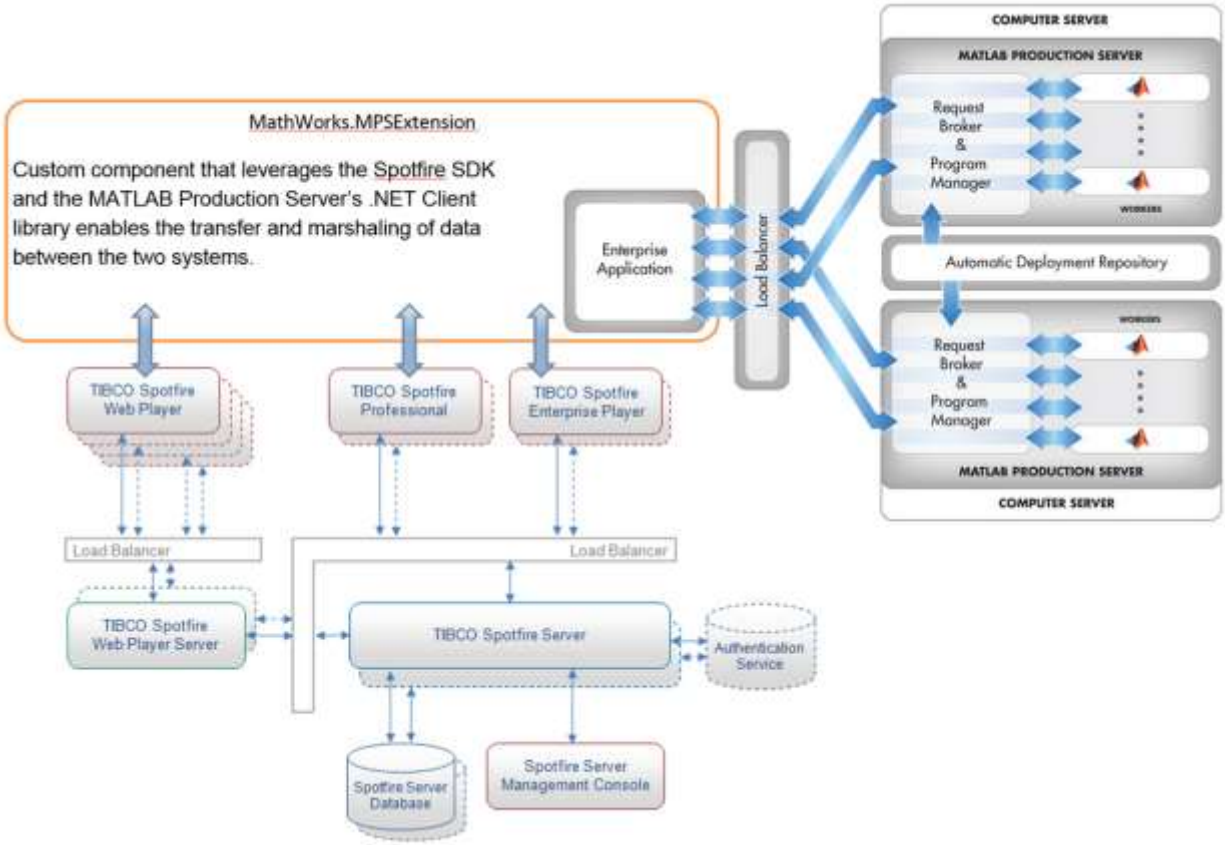
MATLAB Production Server in a clustered configuration

To service more requests and optimize response time, system administrators can deploy additional instances of MATLAB Production Server. MATLAB Production Server uses a stateless architecture, which enables any free worker in a pool to service a request. Client requests can be directed to any MATLAB Production Server in a cluster using load balancing either built within your application or as an external device. This approach creates a system architecture that is resilient to failures.



Additionally, to support ease of deployment, packaged MATLAB programs can be published in a repository for automatic deployment. By creating a shared repository, system administrators can automatically deploy MATLAB programs across multiple instances of MATLAB Production Server.

Together this resulting setup looks like:



Appendix D: Modify configuration files

In case the user needs to modify the contents of the ServerConfiguration.xml (to point to different servers) or FunctionList.xml (to modify function presets) the user will need to either:

a) Modify the file locally (this will change only the settings on the local user's machine)
b) Modify the .SPK file and re-deploy to the server (making it available to ALL users)

To modify the files locally, please edit the files at:

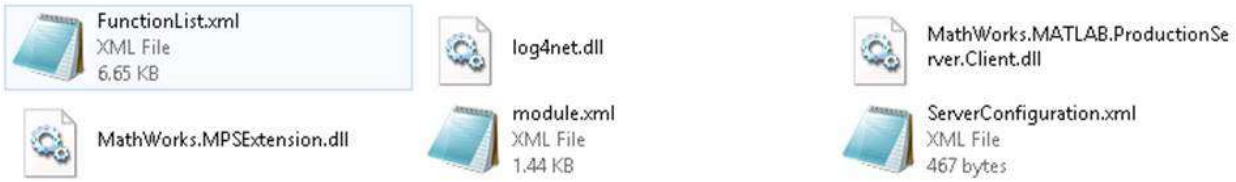
C:\Program Files (x86)\TIBCO\Spotfire\<Spotfire version>\Modules\MathWorks.MPSExtension_<VERSION>

To modify the .SPK file, users will need to build a new Spotfire Package. To accomplish this,

1. Rename the .spk file to .zip and unzip the contents.

In the Contents folder, you will find a cabinet file:

Extract the cabinet file to a clean working folder:



2. Edit the XML files as necessary and then build and redeploy the Spotfire package as in Appendix B.

Appendix E: MATLAB Function Service Discovery

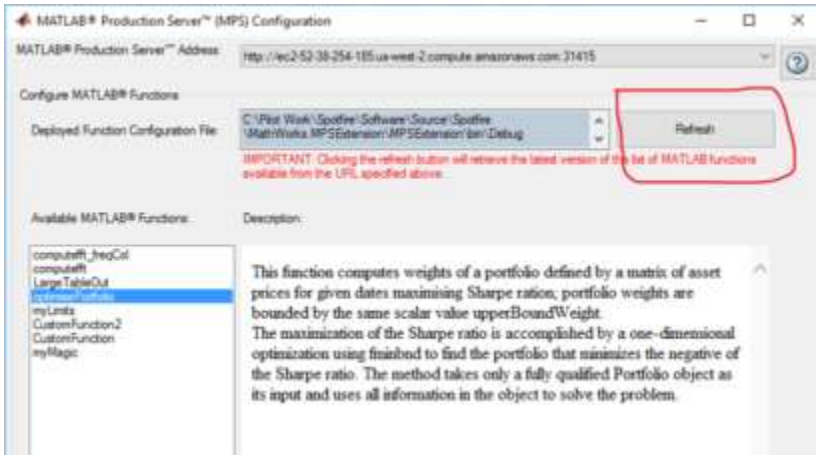
It is possible for the MATLAB developer to enable 'service discovery' of the MATLAB functions that have been deployed to MATLAB Production Server. When the MATLAB developer adds or removes MATLAB functions to MATLAB Production Server, the updated FunctionList.xml file needs to be provided to the Spotfire end user. This ensures that the Spotfire user is aware of the latest functions available, and is not making calls to functions that could have been removed. The service discovery feature allows the Spotfire user to query and automatically download the FunctionList.xml file, keeping the end users version in sync with the MATLAB functions available in MATLAB Production Server.

To enable this feature, package the below MATLAB code and the updated FunctionList.xml file into a '.ctf' archive called **RefreshFunctionList** using MATLAB Compiler SDK. The MATLAB function is as shown below.

```
function [ output_args ] = RefreshFunctionList()
%REFRESHFUNCTIONLIST Summary of this function goes here
% PARSEXML Convert XML file to a MATLAB structure. Change path to
%XML file if different
filename = 'FunctionList.xml'
output_args = fileread(filename);
```

Deploy the archive file to MATLAB Production Server.

In the Spotfire application, when the user clicks on the 'Refresh' button in the MATLAB Production Server extension tool, it calls the RefreshFunctionList function and updates the FunctionList.xml file. Please make sure that the server address for MATLAB Production Server is correct before clicking the button.



Close the UI and open the MATLAB Production Server tool from the menu options to see the latest updates.

Appendix F: Troubleshooting

1. **Static analysis during the deployment of the extension:**

The *MathWorks.MPSExtension* component relies on the .NET client library for the MATLAB Production Server. This client library in-turn relies on the Google Protocol Buffer DLL that is packaged as a part of the client library into the same DLL. On deployment validation, the Spotfire tool checks dependencies and since the Protocol Buffers DLL is embedded in the client library, it is unable to locate it and will not validate. To fix the issue the Protocol buffers is packaged as a separate component that needs to be installed to create a valid configuration.

2. **Web Player does not work with the extension:**

Updates to the *MathWorks.MPSExtension* component (such as when rebuilt) can be pushed to the Spotfire Server which in turn pushes the component to each desktop client. The Web Player however needs to be updated manually. This is done by modifying the *Spotfire.Dxp.WebUpgradeTool.exe* tool. A description of how to do this can be found in the *Installation_Instructions.docx* document.

Appendix G: Code Listings

Listing A: MATLAB Demo code for Portfolio Optimization

```
function weights = optimisePortfolio(dates, prices, upperBoundWt)
% OPTIMISEPORTFOLIO Simple portfolio optimization
% This demo computes weights of a portfolio defined by a matrix
of asset
% prices for given dates maximizing Sharpe ratio;
%
% The portfolio weights are bounded by the same scalar value
upperBoundWeight

%% Check inputs
dates = dates(:);
validateattributes(prices, {'numeric'}, {'2d'});
validateattributes(upperBoundWt, {'numeric'}, {'scalar'});
assert(size(prices, 1) == length(dates), ...
    'optimisePortfolio: Size mismatch; Expected number of rows
in prices is %d and received is %d', ...
    length(dates), size(prices, 1));

%% Calculate Returns
% Calculate daily returns
assetReturns = tick2ret(prices, dates, 'Continuous');  % log
returns, Financial TB

%% Setup portfolio
p = Portfolio;
p = p.estimateAssetMoments(assetReturns); %Mean and covariance
of asset returns from return data.
p = p.setDefaultConstraints; %Non-negative weights that must sum
to 1.
p = p.setBounds(p.LowerBound, upperBoundWt*ones(1, size(prices,
2)));

%% Find min variance portfolio
weights = p.estimateMaxSharpeRatio();
```

Listing B: Android Application Source

MainActivity.java

```
package com.mathworks.demoapp;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
```

```
import android.view.Menu;
import android.view.MenuItem;
import android.webkit.WebSettings;
import android.webkit.WebView;


public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        /* Create and load our analysis */
        WebView myWebView = (WebView) findViewById(R.id.webview);
        myWebView.loadUrl("http://spotfire-00-
ah.mathworks.com/SpotfireWeb/ViewAnalysis.aspx?file=/Demo/Portfo
lio%20Optimization%20Example%20Mobile%20Layout&configurationBloc
k=SetPage%28pageIndex%3D0%29%3B&options=7-0,8-0,9-0,10-0,11-
0,12-0,13-0,14-0,1-0,2-0,3-0,4-0,5-0,6-0,15-0");

        /* Enable Javascript support */
        WebSettings webSettings = myWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar
will
        // automatically handle clicks on the Home/Up button, so
long
        // as you specify a parent activity in
AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
```

```
    }
}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mathworks.demoapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.INTERNET" />

</manifest>
```

Appendix H: Class diagram for MathWorks.MPSExtension