SpreadServe Documentation

Release 0.2.0

John O'Sullivan

May 25, 2017

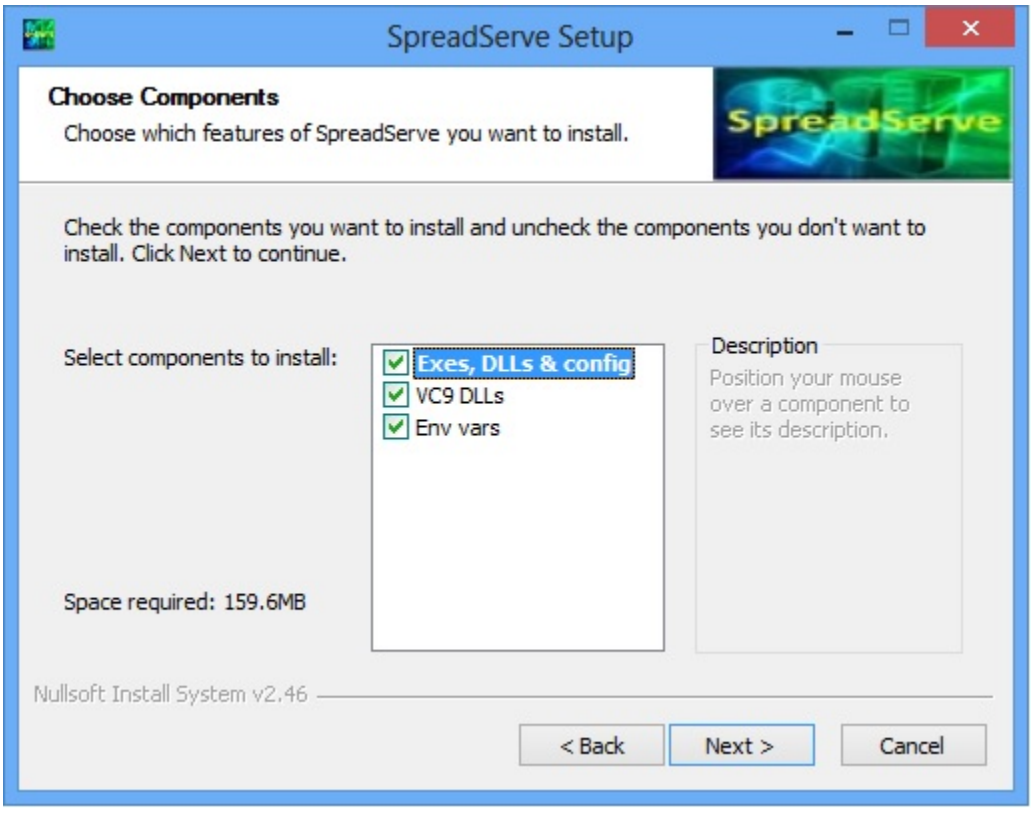
Contents

Contents:

CHAPTER 1

The SpreadServe Installer

SpreadServe is packaged as an NSIS installer. The install process is designed to make both installation and repackaging simple and modular, and to minimise system wide host impact.



Installing SpreadServe for the first time

- Download the latest installer from http://spreadserve.com/s3/downloads.html to your target Windows host.
- Launch the installer by double clicking, or running it at the command line.

- Check the license terms and click "I Agree".
- Choose your install options
  - Exes, DLLs & config: this will copy all the SpreadServe files into `c:\SpreadServe`. You can change the install directory later in the process.
  - VC9 DLLs: this will install Microsoft's Visual C++ 9 DLLs. You must do this the first time you install SpreadServe if you do not already have the VC9 CRT DLLs installed. If you do already have them this option will still be harmless.
  - Env vars: this will set two system wide environment variables; `SSROOT` and `SSROOTX`. You should do this the first time you install SpreadServe, unless you want to set the variables yourself, perhaps as user variables rather than system wide.
- Click Close when the Installer finishes.
- Open a new command shell, cd to `%SSROOT%\sh`
- `launch SIT baseweb`
- Point your browser at http://localhost:8090 to see the web UI home page.

Uninstalling SpreadServe

- Delete the `%SSROOT%` directory tree
- Go to Control Panel/System/Advanced System Settings/Environment Variables and delete the SSROOT and SSROOTX variables from System variables.
- That's it! SpreadServe does not touch the Windows Registry.

Directories and environment variables

By default SpreadServe installs in `c:\SpreadServe\ss<version>`. You can change this in the install process, but below we assume you've taken the default. Where relative directories are mentioned, without the full path name, they are relative to the `SSROOT` environment variable which SpreadServe requires to be set to the root of it's install. If you selected the 'Env vars' option in the install you won't need to set the environment variables yourself. If not, you can set them in a command shell like so:

```
set SSROOT=c:\SpreadServe\ss0.4.2
set SSROOTX=c:/SpreadServe/ss0.4.2
```

`SSROOTX` is the same as `SSROOT`, but with backslashes translated to forward slashes, and any spaces rendered as %20.

Installing SpreadServe under `c:\Program Files` can cause problems for some Addins: see Constraints for more detail.

Repackaging SpreadServe

SpreadServe's install process has been designed for repackaging in corporate deployment systems. The first install option, for Exes, DLLs & config, will create a SpreadServe directory tree and nothing more. It does not touch the registry, and does not install any DLLs under `c:\Windows`. The SpreadServe directory tree can be zipped up and unzipped on other hosts. SpreadServe doesn't care which directory it lives in; it determines that at run time from the SSROOT and SSROOTX environment variables. The second and third install options, for VC9 DLLs and environment variables can be omitted on subsequent installations, or in repackingings where alternate mechanisms for environment variables and system DLLs are appropriate.

CHAPTER 2

SpreadServe User Guide

Starting and stopping SpreadServe

Once you've installed SpreadServe, you can start and stop it's server processes using a couple of scripts in the sh subdirectory. Here's how you would start SpreadServe:

```
cd %SSROOT%\sh
launch SIT baseweb
```

And here's how you can halt SpreadServe:

```
cd %SSROOT%\sh
halt SIT
```

The launch script takes two parameters: environment name (SIT), and profile (baseweb). SpreadServe processes use the environment name to identify each other when they communicate. A SpreadServe deployment used for testing might have an environment name of SIT or UAT, and one used for hosting production services might be PRD. Environment names allow multiple instances of SpreadServe to run on the same host. The second parameter, for profile, identifies a file in the the cfg subdirectory. The profile files are in JSON format, and specify which processes SpreadServe should start when launched; baseweb.jsn specifies a default minimum set of processes.

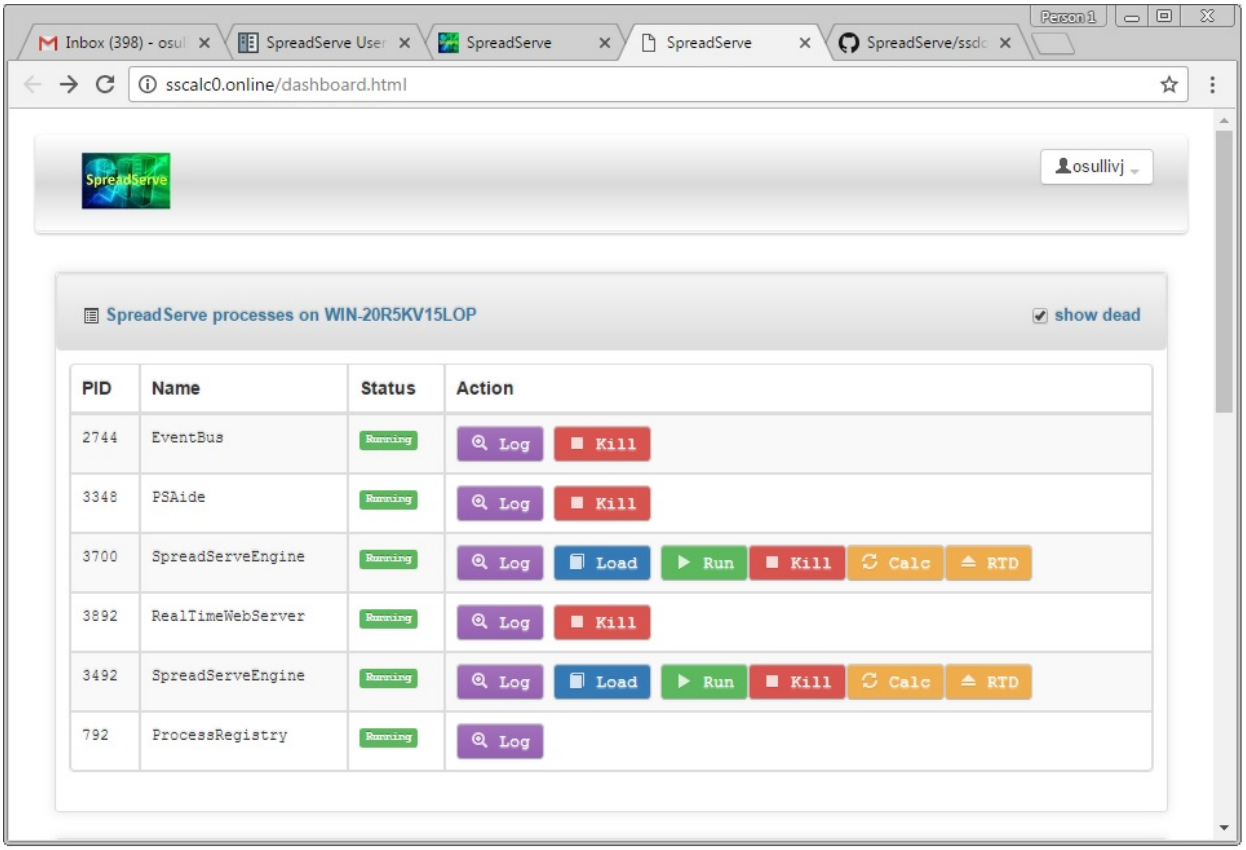
Logging in to the web user interface

When SpreadServe is running, you can point your browser at the SpreadServe RealTimeWebServer and login by clicking the login button at top right.

Note the port number in the URL: 8090. So if SpreadServe is running on a Windows Server with the hostname sshost, the URL you should enter in your browser will be http://sshost:8090 To login you should use your Google or Github identities.

User Interface: dashboard

When you've logged in you should see a page like this in your browser...

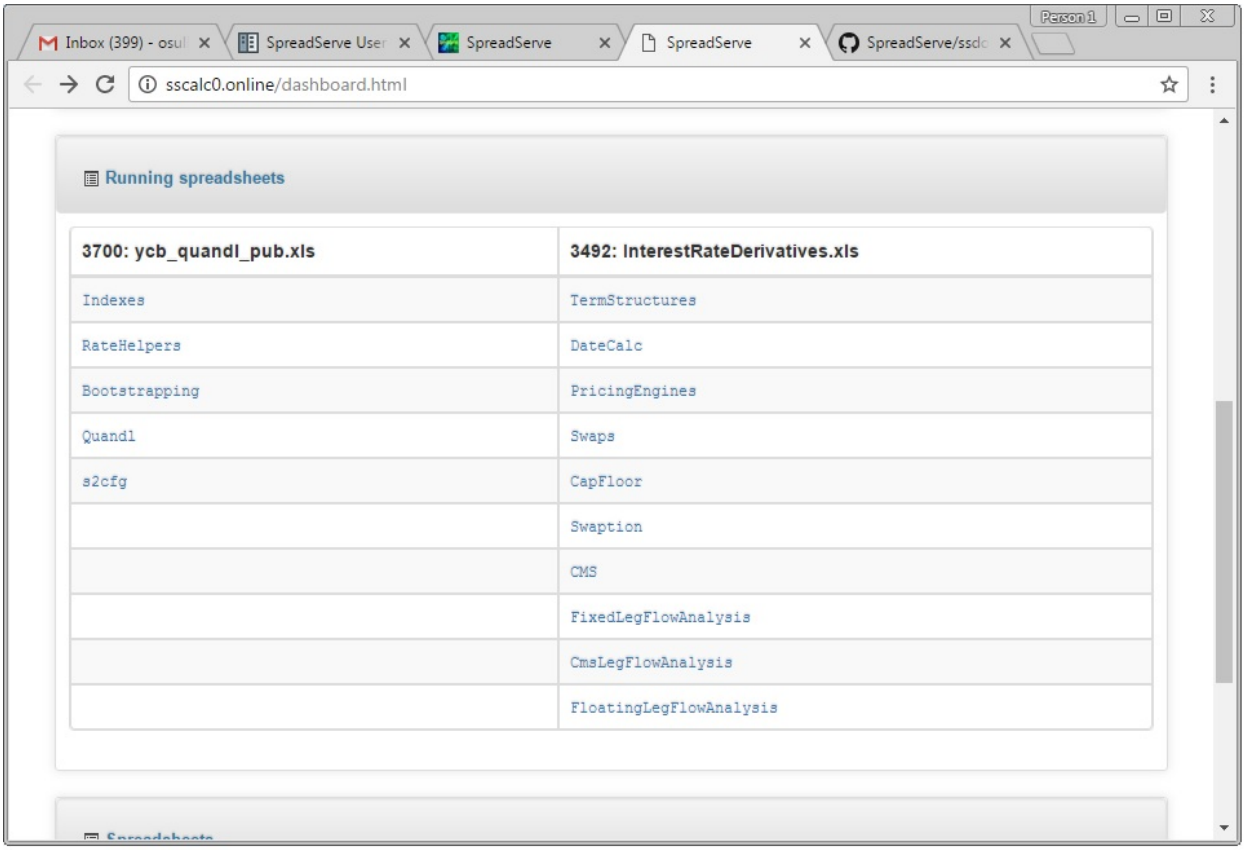


This is the SpreadServe dashboard. It's used for starting and stopping SpreadServe processes, examining their log files and loading spreadsheets into running instances of the SpreadServeEngine. The top table in the page gives the process ID of each process, its name, its status, and a set of buttons for operations. You can use the process ID to lookup the process in Windows Task Manager if you're logged on to the server hosting SpreadServe.

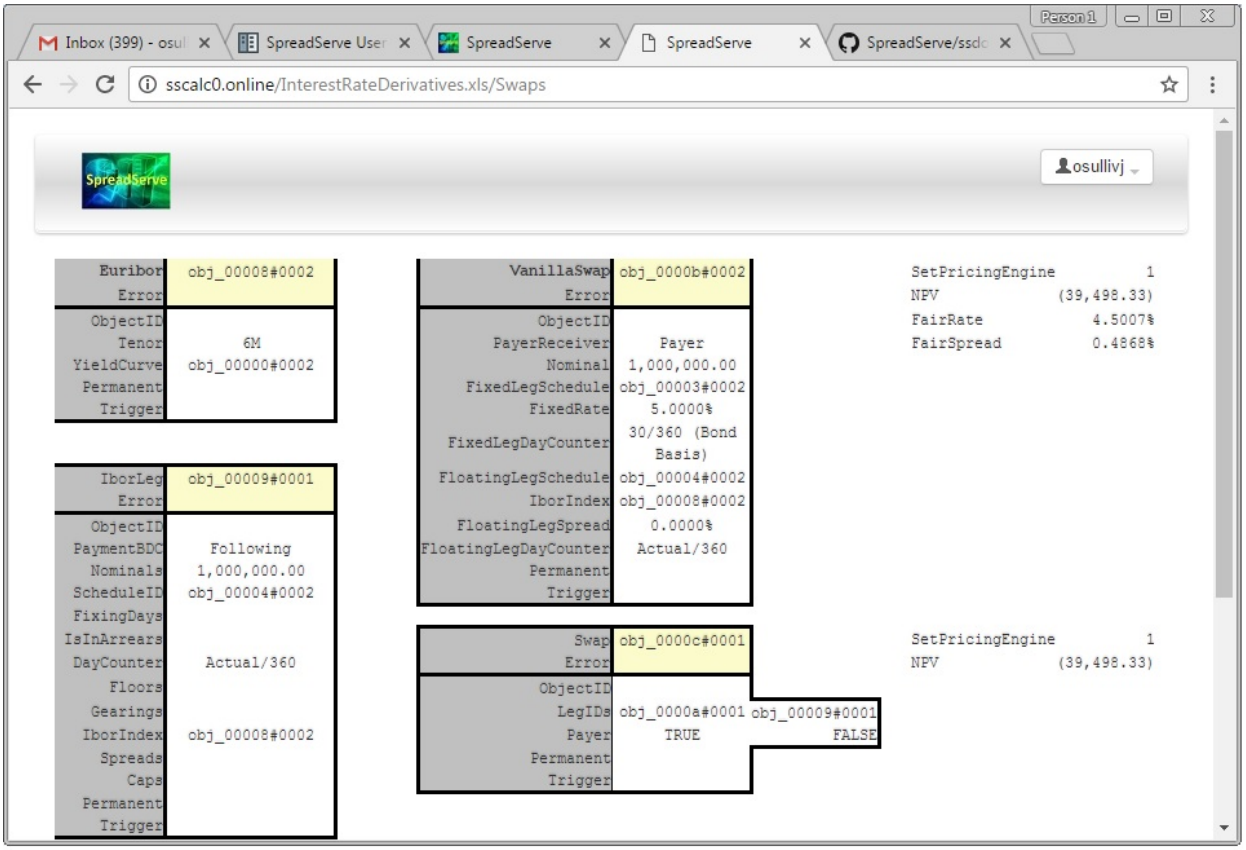
- Log: clicking this button for any process will show you the current log file for the process. The file will be in the %TEMP% directory. The file is displayed 'raw' in the browser, so you'll need to use the back button to return to the dashboard page.
- Kill: in normal operations you should only use this to kill of instances of SpreadServeEngine.
- Run: clicking the Run button next to SpreadServeEngine will create another engine instance. You need one instance of the SpreadServeEngine for each spreadsheet you're running.
- Load: instruct a SpreadServeEngine to load a spreadsheet from the repository. A dropdown list of all the spreadsheets in the repository will appear, and you can select on for loading. Then click on Live Sheets to confirm the spreadsheet has loaded. If the engine was already running a sheet, that original sheet will be unloaded before the new one loads up. See below for more detail on Live Sheets and the Repository.
- Calc: instruct a SpreadServeEngine to recalculate a sheet. Equivalent to doing shift-ctrl-alt-F9 in Excel.
- RTD: invoke ServerTerminate on any RTD servers that are loaded.

At the top right of the table of processes the 'show dead' check box hides or reveals a complete list of processes that includes dead or not yet started processes.

User Interface: Live Sheets If you use the SpreadServeEngine Load button to load the InterestRateDerivatives.xls spreadsheet, then scroll down to the Running Sheets table, you should see something like this...

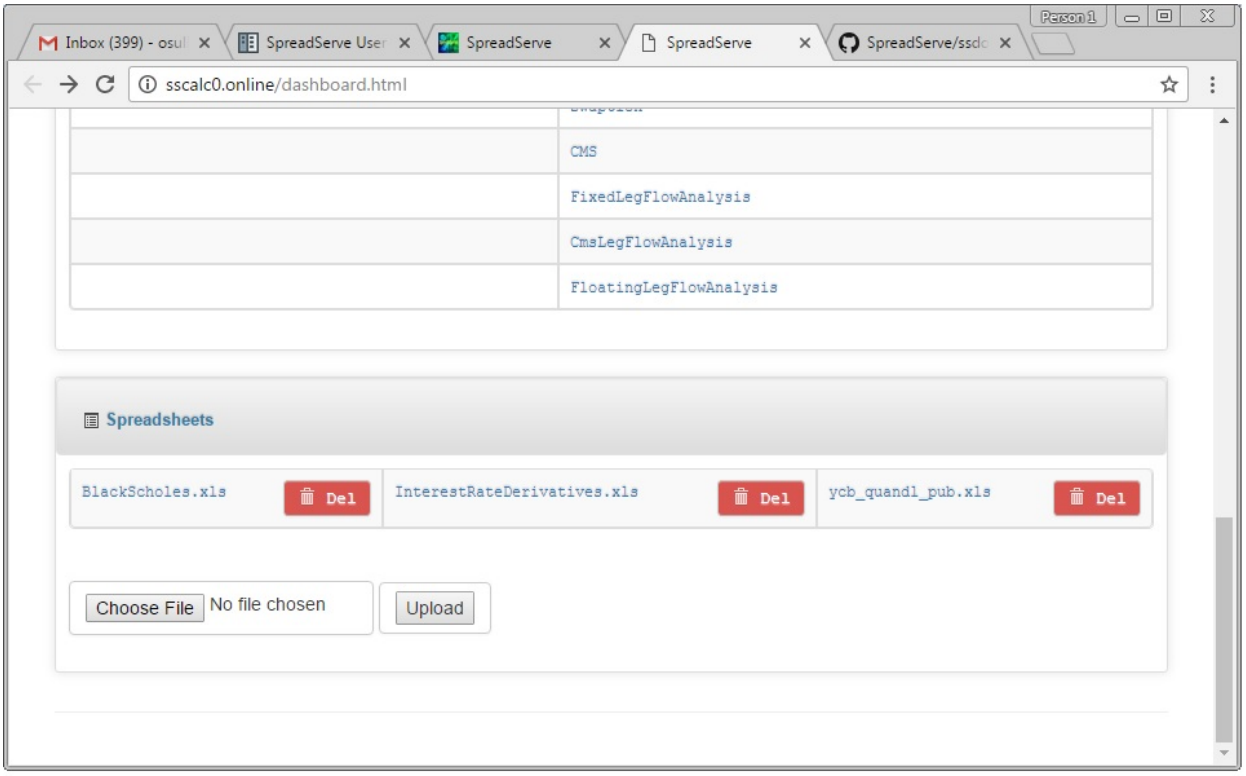


Naturally, all the live sheets within a live spreadsheet are clickable links that will take you through to a view of that sheet. Try clicking on Swaps under InterestRateDerivatives.xls and you should see this...



This is just what you'd see in Microsoft Excel if you loaded the same SpreadSheet. Try it and see for yourself. You can find the spreadsheet in the SpreadServe directory tree at `%SSROOT%\py\http\repo\InterestRateDerivatives.xls`. Now in Excel all the cells are editable, both those with formulae, and those holding raw data. You can change the cells and see the spreadsheet recalc. In the SpreadServe GUI you can change the raw data cells, but you can't edit the formula cells. In the SpreadServe GUI try clicking on any of the cells that have the dot dot dot underlining. You'll see a pop up that will allow you to change the data. If you do, the spreadsheet will recalc and you'll see the changed values in your browser, just as you would in Excel. But with SpreadServe, there could be any number of browsers looking at that live sheet, and they'd all see the same data, and the same recalculation.

User Interface: Repository On the dashboard page, scroll right down to the last table. You'll see a list of the spreadsheets available for loading with the Load button on the dashboard page...



The repository table also allows you to upload spreadsheets from your local file system. Hit the browse button to select a spreadsheet. When you've OKed the file selection dialog, hit the Upload button to persist the spreadsheet in the repository directory on the server. Next time you hit the load button your spreadsheet will be listed. **Constraints**

- Limited VBA support. SpreadServe does not support XLAMs. Only embedded VBA can be used. Range can't be used as parameter type. Parameters should be built in types only: ints, floats, strings.
- Processes run with elevated permissions access the registry differently, which can prevent Excel-DNA based addins from working. Elevated permissions are required to run binaries installed under `c:\Program Files`. However, processes with elevated permissions can't access per user settings under `HKEY_CURRENT_USER`, instead they'll go to `HKEY_LOCAL_MACHINE`. This is by design to prevent exploits. Excel-DNA puts its auto registry entries under `HKEY_CURRENT_USER` for RTD registration. See ExcelRtd.cs. Consequently, to use addins with Excel-DNA based RTD servers, we must install SpreadServe outside `c:\Program Files`, so we can run it without elevated privileges. http://stackoverflow.com/questions/5649544/component-creation-fails-under-uac-admin-works-without-uac-elevation https://msdn.microsoft.com/en-us/library/bb756926.aspx
- In a fully automated server environment one can't replicate the human driver of a spreadsheet. For instance an user will give a sheet time to complete a long calc, then if they see #VALUE or #NUM, may hit F9 again. That use pattern can occur in sheets with hidden dependencies that are not on the graph, eg YieldCurveBootstrapping and the Rate/Date column references to an object ID that doesn't change on a calc driven only by changed inputs. A complete recalculation of all nodes on the graph, irrespective of changed inputs, is necessary to force the changes through. But if a sheet is indirectly circular because it's doing quandl queries and notifying results via RTD, a complete recalc on every RTD update causes never ending computation. We have a configuration setting in sseng.ini to control what kind of recalculation is done on an RTD update: `RTD_FULL_CALC`. You should only set this when there are no circular dependencies, hidden or otherwise.

Connectors: getting data in and out of SpreadServeEngines

Earlier we saw how to start and stop SpreadServe, load a spreadsheet into a SpreadServeEngine, and view the spreadsheet in a browser. We also saw how we can edit the sheet inputs from a browser, prompting the sheet to recalculate, and then see the results display in every browser looking at that live sheet. Recall also, that SpreadServe is about bringing resilience, automation and scalability to spreadsheets. Resilience comes from being in a monitored and managed server environment. Scalability comes from the ability to spin up as many SpreadServe engines as you need, and to have many browsers viewing the same sheet. So where does automation come into the picture? The answer to that question leads us to SpreadServe's input and output connectivity. As we saw, SpreadServe can take input from web pages, and it can display loaded sheets in a browser. SpreadServe can also connect to databases and messaging systems for input and output. Two sample connectors are supplied with SpreadServe: BlackScholesMockMarketData and dblog.

BlackScholesMockMarketData

One of the sample connector programs supplied with SpreadServe injects mock market data into the Black Scholes spreadsheet. You can run BlackScholesMockMarketData like this:

```
cd %SSROOT%\py\smpl
..\..\sh\sspy black_scholes_mock_mkt_data.py -ENV SIT
```

Note that you may need to change the path and environment name to match your install. Once BlackScholesMockMarketData is running, take a look at the BlackScholes.xls sheet in Live Sheets. You'll see the stock price ticking up and down, and the whole sheet recalculating. This sample is designed to illustrate an investment banking use case: how a pricing sheet developed by a trader can be taken off the desktop, automated, and shared with many colleagues. A real implementation would have a far more complex spreadsheet, and take its data from a real time market data system.

dblog

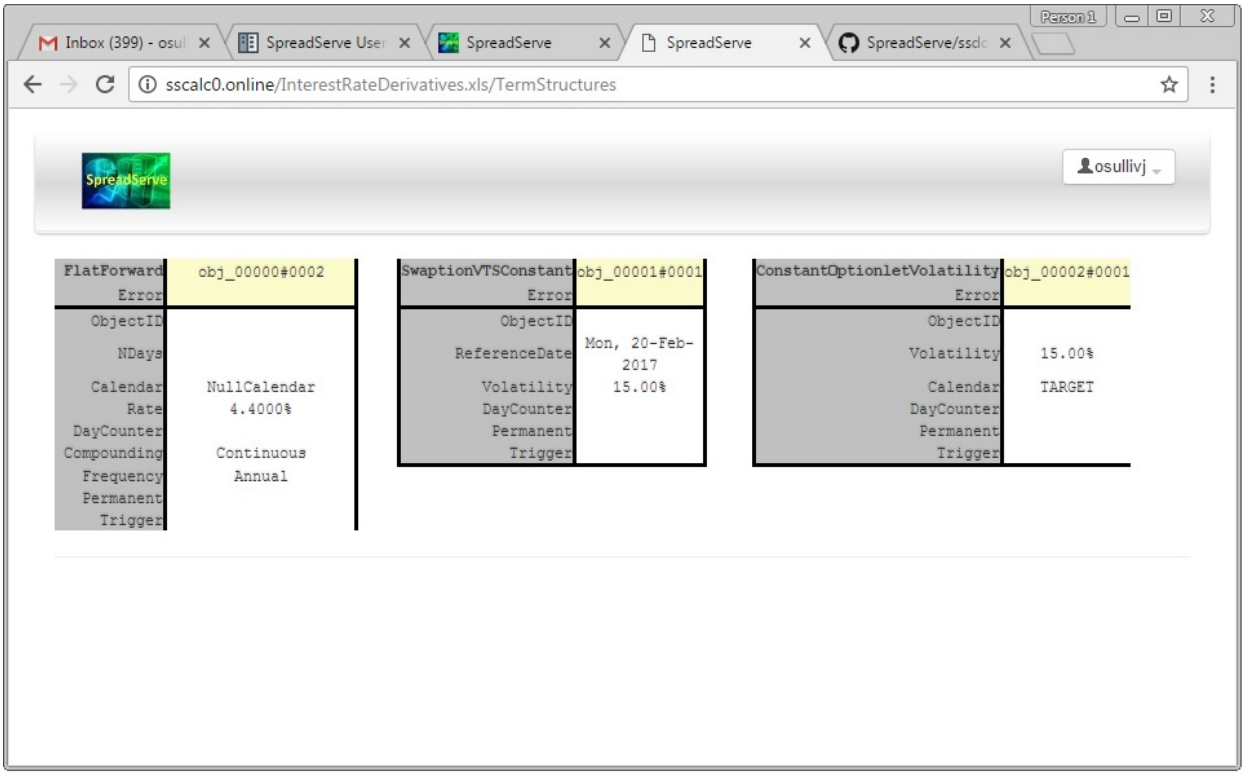
dblog consists of two processes; one coded in Java, for JDBC DB connectivity, and one in Python, built on SpreadServe's SocketServer implementation. Here's how you launch the Python SocketServer part of dblog:

```
cd %SSROOT%\py\sock
..\..\sh\sspy dblog.py -ENV SIT
```

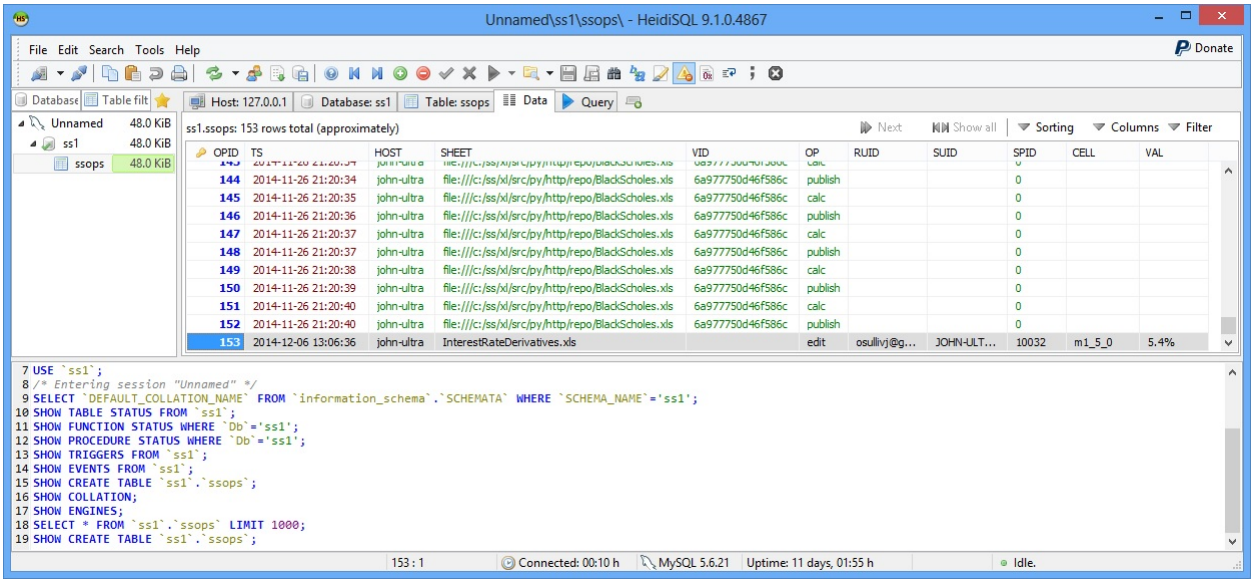
And this is how you launch the Java process:

```
cd %SSROOT%\sh
dbconn
```

There are several configuration dependencies here, and this will only work out of the box if you have a MySQL install on your SpreadServe host. We'll detail the config below. Assuming your config is correct you'll see operations tracked in the database. Try loading the InterestRateDerivatives.xls spreadsheet, navigate to the TermStructures sheet, and change the Rate cell from 4.4% to 5.4%.



Obviously the dependent sheets and cells will recalc. Look at the NPV on the Swaps sheet for instance. You'll also find that the SSOPS table in the database has recorded the change to the sheet too. At SpreadServe we like the HeidiSQL DB clients. Here's how it looks to us...



Notice how that last row in the SSOPS table records the timestamp, the spreadsheet, the operation (edit), the user ID, and the cell that was changed (m1_5_0) and the new value (5.4%).

dblog configuration

The configuration for the dblog Python SocketServer process is in `%SSROOT%\cfg\dbcfg.py`, and the Java process gets its config from `%SSROOT%\cfg\dbconn.props`. The Python variables in dbcfg.py should match your DB schema, and in dbconn.props the connection details must match your JDBC driver and DB connection. If you're not using MySQL, you'll need to add the relevant JDBC driver to the lib directory, and fix the CLASSPATH setting in shdbconn.cmd to pick up the jar.

XLL configuration

To add an XLL to your SpreadServe deployment you need to edit `cfg\xll.txt`. In the standard install it looks like this:

```
file:///c:/SpreadServe/ss0.4.2/bin/xlcall32.dll;cdecl;refreshdata
file:///c:/SpreadServe/ss0.4.2/bin/quantlibxl-vc110-mt-s-1_4_0.xll;cdecl;refreshdata
file:///c:/SpreadServe/ss0.4.2/bin/SSAddin.xll;stdcall;refreshdata
```

To add your XLL, copy it to the `%SSROOT%\bin` directory, then add another line to xll.txt modelled on the the lines that reference xlcall32.dll and the QuantLib XLL. Note that each line has three parts separated by semicolons. Firstly the path to the XLL, then the calling convention, and finally an RTD switch. The path must be explicit and absolute, you can't use the `SSROOT` environment variable. The calling convention should be `cdecl` or `stdcall`; XLLs implemented in C++ will probably be `cdecl`, and those in C# `stdcall`. However this is not a hard and fast rule, and if you're not sure which calling convention your XLL uses then examine it with dumpbin or depends and look at the exported symbols. This article by the immortal Raymond Chen will enable you to determine whether you're seeing symbols using stdcall or cdecl. The third part of the line will be *refreshdata* or *norefreshdata*. It should always be set to the former unless you're using an XLL which generates RTD updates and you want to disable them.

CHAPTER 3

SpreadServe Configuration

SpreadServeEngine configuration

`sseng.ini`: this config file lives in `%SSROOT%\bin` as it needs to be in the same directory as sseng.exe. sseng.exe uses it to determine the location of various resources including dynamically loaded DLLs and the xll.txt file that specifies the XLLs to be loaded. Most of the settings are purely internal; two may be of interest to users.

- `LICENSE_FILE`: path to the license key file, which can contain an offline key to stop the SpreadServeEngine phoning home to spreadserve.com.
- `DNS_HOST_NAME`: path to a flat text file containing a hostname. If the file exists, SpreadServeEngine will use this hostname when phoning home to spreadserve.com.
- `OWNER`: the email address associated with this SpreadServe host by spreadserve.com.
- `FORMAT_LOADING`: switches cell formatting on or off to accelerate loading xlsx files. No impact on xls handing.
- `XLL_CFG_FILE`: path to the file that specifies the XLLs to be loaded.
- `XLL_REG_FILE`: path to the file sseng.exe uses to dump signatures of XLL worksheet functions that have been successfully registered.

`dns_host_name.txt`: if this file exists in `%SSROOT%\cfg` then SpreadServeEngine will use its contents as hostname in handshakes, ping and load messages when phoning home to spreadserve.com. RealTimeWebServer looks for dns_host_name.txt too. You should include the port number too if RealTimeWebServer isn't running on port 80. In an enterprise environment you might put something like `mydesktoppc.intranet:8090` in dns_host_name.txt.

RealTimeWebServer configuration

The RealTimeWebServer implementation is mostly in one Python module: `%SSROOT%\py\http\rtwebsvr.py`, which has gets configuration variables from `%SSROOT%\cfg\webcfg.py`. If you want to switch SpreadServe to work with Active Directory authentication, rather than it's default social login, then edit webcfg.py to configure user group mappings. You'll also need to supply the `AUTH` command line parameter to RealTimeWebServer. See the command line parameters section below for detail on `AUTH`.

- `ADGroupMappings`: a dictionary defining the Active Directory groups that for which user membership will give view, edit or admin permissions.

  - `view` defaults to `Users`. All Windows hosts have a `Users` group. `view` gives a user basic view only privileges.
  - `edit` defaults to `SSEdit`. You probably won't have an SSEdit group in your Active directory group mappings. You should either create one, or change the edit setting to name an appropriate existing AD group. Users with edit permissions can click on a sheet's value fields in a browser and change them. They can also hit the Calc and RTD buttons for a SpreadServeEngine on the dashboard.
  - `admin`: defaults to `SSAdmin`. As with SSEdit, you should either create an SSAdmin group in your environment, or map admin to an appropriate existing AD group. Users with admin permission can start and stop SpreadServe processes via the dashboard, and upload new spreadsheets to the repository via the repository page.

- `RTWSPort`: defaults to port 8090. Change this to 80 if you want to run RealTimeWebServer on the default HTTP port. If you do, check no other process is has taken port 80, like IIS. Also ensure the user ID running the process has enough rights to use port 80.
- `ApiKey`: REST API clients should supply this in their Authorization headers. Set to `None` to shut off API access, or change to your own secret value to control access.

SpreadServe command line parameters

All SpreadServe processes, whether they are RealTimeWebServer, SpreadServeEngine, Dora, Pan, SocketServer or DBLogServer take a common set of command line paramters. Some have custom parameters that tailor a specific part of their behaviour. You'll see the parameters used in the shell scripts in `%SSROOT%\sh` and the JSON launch files in `%SSROOT%\cfg`. Command line options are always presented with a leading hyphen and a following value eg `-ENV SIT -NAME DBLogServer`.

- `HTTP_PORT`: supply this on the rtwebsvr.py command line to change the RealTimeWebServer port. For example `-HTTP_PORT 80` to run on port 80.
- `AUTH`: supply this on the rtwebsvr.py command line to specify the authentication mechanism. The three possible settings are
  - `sscld`: the default. Cloud authentication with spreadserve.com. To edit permissions at http://spreadserve.com/adm/cldperms.html you must ensure your email address is set in `sseng.ini:OWNER`
  - `ssad`: Active Directory. Configure your user group mappings as describe above.
  - `ssna`: No authentication. We suggest you only use this is development and test environments, and not production!
- `ENV`: Mandatory. Environment that this SpreadServe process belongs to. Several environments can co-exist on the same host as components will only recognise and communicate with components in the same named environment.
- `NAME`: Optional. C++ processes will default to the exe name on the dashboard page, and Python processes will show up as `python`. Using NAME you can override these defaults to make processes more readily identifiable.
- SpreadServeEngine: these options are specific to SpreadServeEngine
  - `PIPE_LOG`: used to switch on detailed logging from the engine's subsystems. For instance `-PIPE_LOG BSX`. The value of the option should be one or more letters from D, B, S, X. Switching some or all of these options on will generate a lot of logging.
    - `D`: general debugging output.
    - `S`: spreadsheet compiler and interpreter
    - `X`: XLL subsystem
    - `B`: Basic

  - SUPPRESS_OUTPUT_LOG: set to 1 to switch off logging of the engine's interprocess communication with the RealTimeWebServer. SpreadServeEngine sends whole web pages in HTML as well as JSON formatted updates of sheet state. That can generate a huge amount of logging if there's a high update frequency, and it can make it difficult to follow higher level events like incoming data triggering recalcs.

SpreadServe scripts

The %SSROOT%\sh directory holds several scripts for starting and stopping SpreadServe processes singly or as a group.

- launch.cmd: launch a group of SpreadServe processes.
- halt.cmd: stop a group of SpreadServe processes.
- sseng.cmd: launch a SpreadServeEngine process.
- sspy.cmd: launch a SpreadServe Python process.
- dbconn.cmd: launch the DB connector.

See the User Guide for examples of their use.

Log files

SpreadServe log files appear in the %TEMP% directory.

Profiles

Profiles are JSON config files that determine which SpreadServe processes are launched at startup time. A launch command for SpreadServe takes the form launch <environment> <profile>. for instance launch SIT baseweb. Several ready made profiles are supplied in the %SSROOT%\cfg directory. They are...

- pandora: a minimal profile that only starts ssdora.exe, the ProcessRegistry and sspan.exe, the EventBus. Useful for developers who want to control hte launch of other processes, perhaps because they're debugging.
- base: launches three processes: ProcessRegistry, EventBus and sseng.exe, the SpreadServeEngine itself.
- baseweb: launches the same three processes as base, but with the addition of the RealTimeWebServer.
- demo: same as baseweb, but adds BlackScholesMockMarketData to pump fake market data into the BlackScholes.xls example sheet.

Windows Service

The launch.cmd and halt.cmd scripts described above are appropriate for manually launching and halting SpreadServe. You may also find them convenient for other job control systems like AutoSys. You can also configure SpreadServe to run as a Windows Service:

```
cd %SSROOT%\py\util
..\..\sh\sspy windows_service.py install
```

Then you can use Windows' Services GUI to configure Automatic or Manual startup, and to start and stop the service. We recommend you do not use the Local System account to run SpreadServe as a Windows Service, and instead configure it to run under Administrator or some other user account. SpreadServe's RTD capabilities, as implemented in SSAddin, rely on Registry ClassId and ProgId lookup that access the HKCU hive, and they don't wotk under Local System. Once you've created the service you can start and stop SpreadServe at the command line like so:

```
sc start SpreadServe
sc stop SpreadServe
```

To automate SpreadServe start and stop times on a specific host you can use Windows Task Scheduler to invoke sc start SpreadServe and sc stop SpreadServe.

CHAPTER 4

SpreadServe Tech Spec

Technical specification

SpreadServe is a Windows Server hosted server product with a web UI.

- OSes: any Windows desktop OS from Windows 7, and any server OS from 2003. SpreadServe has been tested on Windows 7, Windows 8, Windows Server 2008 & 2012.
- User interface: web UI built with Angular 1.5, <http://Bootstrap 3.3 and jQuery 2.2 enabling job control and spreadsheet interaction from a browser.
- APIs
  - Python: the best API for Web integrations, as well as rapid, ad hoc development. SpreadServe uses Python 2.7.8, 3.x is not yet supported.
  - C++: the best API for high performance systems integration. For example, this is the right way to connect SpreadServe to a low latency pub sub messaging system like TibRV or Solace. Integrations should be built with MS Visual Studio 2008 or later.
  - REST: external processes can drive SpreadServeEngines via the RealTimeWebServer REST API.
  - Java: the best option for relational database integration with JDBC. spreadserve.jar is built with JDK 1.6.
- Core implementation: SpreadServe's native implementation language is C++ built as 32 bit binaries with Microsoft Visual C++ 9.
- VBA: SpreadServe supports VBA macros, but does not support VBA GUI operations.
- XLL: SpreadServe supports XLL addins, and has been tested with the QuantLib 1.4.0 addin.
- RTD: SpreadServe supports RTD updates.
- Excel file formats: Excel 97-2003 .xls as well as the default Excel .xlsx format are supported.
- RDBMS: the dblog component enables logging of all SpreadSheet operations to an RDB. SpreadServe has been tested with MySQL 5.6, and should work with any RDB that has a JDBC driver.
- Sockets: The SocketServer enables connections from processes based on other hosts and implementation technologies.

- Open Source Software: SpreadServe uses several powerful OSS technologies as building blocks. They include Tornado, Python, STLport , Boost and Apache Open Office.
- Cloud ready: SpreadServe has been deployed and tested on Amazon EC2 Windows 2008 & 2012 AMIs.

CHAPTER 5

SpreadServe Licensing

Dual license model

SpreadServe is available under two kinds of license: public and offline.

- Public
  - Free: there is no charge for running a public SpreadServe instance as an AMI or as installed via the install kit.
  - Phone home: public SpreadServe instances phone home to spreadserve.com. The SpreadServeEngine sends three kinds of messages back to spreadserve.com
    - Handshake: on startup each SpreadServeEngine handshakes with spreadserve.com. If the handshake fails it will shutdown.
    - Ping: each SpreadServeEngine pings spreadserve.com every 30 seconds. That's why the Live Engines panel on the spreadserve.com dashboard flashes every 30 seconds.
    - Load: every time a SpreadServeEngine loads a spreadsheet it phones home with the name of the spreadsheet and all the formulae in the sheet. All that information then becomes searchable for logged in users in the spreadserve.com dashboard.
  - Default: all install kits and AMIs include a public license key.
- Offline
  - Chargeable: there's a monthly fee for an offline license.
  - Silent: SpreadServe does not phone home to spreadserve.com when running with an offline license key.
  - Time bound: offline license keys can be bought in monthly increments for months and years ahead.
  - Host bound: offline license keys are locked to the MAC address of your host.
  - Tiered: the license charge scales with the feature set as you add XLL, VBA and RTD capabilities.
- Private
  - Chargeable: there's a monthly fee for n private license.

  - Quiet: SpreadServe does phone home to spreadserve.com when running with an offline license key, but only for the handshake. There are no pings, and no load information is phoned home. Private hosts do not appear on spreadserve.com's dashboard.
  - Time bound: private license keys can be bought in monthly increments for months and years ahead.
  - Host flexibility: private license keys are not locked to the MAC address of your host. If you have ten licenses, then you can run SpreadServe on any ten hosts.
  - Tiered: the license charge scales with the feature set as you add XLL, VBA and RTD capabilities.

License fee structure

- Choose one of three currencies: GBP, EUR or USD
- RTD
  - Standard Excel formula set with RTD updates.
- VBA
  - Standard Excel formula set with RTD and VBA macros.
- XLL
  - Complete feature set: all Excel formulas with RTD, VBA & XLL support.
- 3pac: DEV, UAT, PRD
  - Three keys for the price of two.
  - Suitable for enterprises running development, UAT and production environments.
- 5pac: DEV, SIT, UAT, STG, PRD
  - Five keys for the price of three.
  - Suitable for enterprises running development, integration, UAT, staging and production environments.

Consultancy

Bespoke development, configuration, and deployment services are provided on a day rate.

Consultancy

Support packages with a preset number of incidents are available.

Beta licensing

Different license terms apply during SpreadServe's Cloud Beta.

- Offline license keys are not chargeable.
- Tiered licensing doesn't apply: all beta licenses grant XLL, VBA & RTD.
- Beta licenses are not host bound; they work on any server.
- Private license keys are not available.

Contact

sales@spreadserve.com

CHAPTER 6

SpreadServe 0.4.2c notes

Install notes

- Log in to spreadserve.com first: when you start SpreadServe on a new server for the first time, SpreadServeEngine will handshake with spreadserve.com and specify its owner using `sseng.ini:OWNER` and its hostname using `%SSROOT%\cfg\dns_host_name.txt` if present. You will only be able to set permissions for your new install via spreadserve.com's Permissions page if you've logged in to spreadserve.com previously with an email address matching the one in `sseng.ini:OWNER`.
- Port 80 or 8090: RealTimeWebServer defaults to 8090. If you stay with this default you should fix `%SSROOT%\cfg\dns_host_name.txt` to be `myhostname:8090`. If you want to run on port 80 make sure no other process like IIS is on 80, that you're running with high enough rights to bind to port 80, and that you fix `%SSROOT%\cfg\webcfg.py:RTWSPort=80`.

Offline license key

Firewall rules may prevent your SpreadServe install phoning home on port 80 to spreadserve.com. Or you may want to avoid sharing any information about commercially sensitive spreadsheets. In those cases you'll need an offline license key, which is free during the SpreadServe beta. The key below is good for any host until 2017-07-31. Paste it into `%SSROOT%\cfg\license.txt`:

```
c3ByZWFkc2VydmUuY29tOjgwLDIwMTcwNzMxLG9mZmxpbmUsRkEtQjEtRUQtREUtQ0EtREUs
MzYyQTYwRkY3RERCMTlBRTQxQkVBRjUyNTZGOTlGQjYyQjlCRUY4NEE0NDM0QTc3QjcyQTMw
QzQ4RUREOUM2NDUzNEMyMDE1ODBGRDM5OTU3NjVGMjUwMzM4RUM4NzM0QTY2RUZEOTEwMTAz
REFGQTU5MzczRTc1RjEyRjIyNjY=
```

CHAPTER 7

SpreadServe Containers I

Introduction and cautionary note

This tech note is a record of the various hacks and tweaks necessary to get SpreadServe 0.3.1 running in a container on an Azure hosted Windows Server 2016 TP3 VM. It's not SpreadServe documentation as such, but it may be useful to any parties interested in running SpreadServe in a container. At the original time of writing - 2015-10-13 - work had progressed to enable a one line launch of SpreadServe containers on spreadserve01.cloudapp.net. There are still outstanding issues to resolve and they will be addressed in a later note. The following Microsoft resources were used as source material for this work.

- *Quick start*: https://msdn.microsoft.com/virtualization/windowscontainers/quick_start/manage_docker
- *Work in progress*: https://msdn.microsoft.com/virtualization/windowscontainers/about/work_in_progress#DockermanagementDockercommandsthatdontworkwithWindowsServerContainers
- *Forums*: https://social.msdn.microsoft.com/Forums/en-US/home?forum=windowscontainers&sort=lastpostdesc&brandIgnore=true&page=7

Stuff that didn't work

I ran into lots of issues when I tried to apply the recipe outlined in *Quick start* to SpreadServe. First up was the simple fact that, by design, you can't run a GUI in PowerShell. MS provide a handy recipe in the Work in progress page for RDPing into a container, which does allow you to run a desktop. It all worked fine for me until the container desktop turned up in my mstsc with a DOS box showing this prompt: `Your account has been disabled. Please see your system administrator` The fix was to do this in a container PowerShell:

```
net user administrator /active:yes
net user administrator <password>
```

This enabled me to RDP in, and run my SpreadServe installer inside the container. My aim was to create a container with SpreadServe installed, and then commit as an image I could reuse. I was able to run the installer in my GUI session, but it appeared to hang. Intially I couldn't figure out why, so I revisited the installer design, which is GUIcentric, and uses NSIS. As well as a SpreadServe directory tree, it sets environment variables, and launches vcredist_x86.exe to install the Visual Studio 2008 C++ runtime. I revised the installer to work in silent mode, to not attempt to set environment variables via the Registry, and to not launch the vcredist_x86.exe installer. Windows Server 2016 TP3 has the Visual Studio 2008 C++ runtime installed already so we don't need it. It took several attempts to rebuild the installer to run cleanly in silent mode. Sys Internals Process Explorer was invaluable in watching the progress of the

installer with the lower pane handles view. If you launch it from a cmd shell, not a PowerShell, in the container host you can see its GUI, which seems to show you all processes, container hosted or not on the VM. Invaluable! You can download procexp.exe to your Azure WM by doing:

```
wget -uri 'https://download.sysinternals.com/files/ProcessExplorer.zip' -OutFile␣
↪ProcessExplorer.zip
```

While I was paring down my install I was attempting to create a SpreadServe container image in two steps. Step one used a dockerfile to create an image with my installer imported:

```
FROM windowsservercore
LABEL Description="SpreadServe for Windows 2016 TP3" Vendor="SpreadServe" Version="0.
↪3.1"
ADD zip/ss031.exe /osullivj/zip/ss031.exe
```

I used that dockerfile to create an ssinstall image in which I'd run the `ss031.exe` installer interactively. Then stop the resulting container and attempt to commit the image:

```
docker build -t ssinstall .
docker run -it --name ss031 ssinstall cmd
cd \osullivj\zip
.\ss031.exe /S
exit
docker commit ef79f92a7479 ss031a
```

This failed everytime at the final step with this error:

```
PS C:\> docker commit ef79f92a7479 ss031a
Error response from daemon: hcsshim::ImportLayer - Win32 API call returned error␣
↪r1=2147549183 err=Catastrophic failure␣
↪layerId=60d6b5ec18466f1b368f67a111bc476b717d9ec497e0097fcf04ff419e01077e flavour=1␣
↪folder=C:\ProgramData\Docker\windowsfilter\60d6b5ec18466f1b368f67a111bc476b717d9ec497e0097fcf04ff4
↪3287807202
PS C:\> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             ␣
↪VIRTUAL SIZE
ssinstall           latest              74e160e024e8        39 hours ago        9.749␣
↪GB
windowsservercore   10.0.10514.0        0d53944cb84d        7 weeks ago         9.697␣
↪GB
windowsservercore   latest              0d53944cb84d        7 weeks ago         9.697␣
↪GB
```

It was only when I stripped down my install process to work completely silently, not touch the registry for environment variable settings, and not attempt to run vcredist_x86.exe that the final commit started working. When this worked I tried extending the dockerfile to make the SpreadServe image build a one shot operation so I wouldn't have to run ss031.exe at the command line. This was the dockerfile:

```
FROM windowsservercore
LABEL Description="SpreadServe for Windows 2016 TP3" Vendor="SpreadServe" Version="0.
↪3.1"
ADD zip/ss031.exe /osullivj/zip/ss031.exe
RUN /osullivj/zip/ss031.exe /S
```

But this would always error at the final step. This is what I got:

```
PS C:\osullivj> docker build -t ss031a .
Sending build context to Docker daemon  55.8 MB
```

```
Step 0 : FROM windowsservercore
 ---> 0d53944cb84d
Step 1 : LABEL Description "SpreadServe for Windows 2016 TP3" Vendor "SpreadServe"␣
↪Version "0.3.1"
 ---> Using cache
 ---> 01e5edd46b4a
Step 2 : ADD zip/ss031.exe /osullivj/zip/ss031.exe
 ---> Using cache
 ---> b8f27700fecb
Step 3 : RUN /osullivj/zip/ss031.exe /S
 ---> Running in 3b05abbb5b2a
The command 'cmd /S /C /osullivj/zip/ss031.exe /S' returned a non-zero code: 1223
```

I'm guessing this must be something to do with the wasy the NSIS installer exits, but can't figure a way to resolve it.

Once I had a working container image committed, my next challenge was port mapping the SpreadServe web server. SpreadServe's RealTimeWebServer listens on port 8888 by default, so I spent a couple of hours tinkering with various permutations of the `docker run` -p parameter. For instance `docker run -p 80:8888`. I also created a firewall rule for 8888 in the container host, since the a post in the forums told me that containers don't have their own firewall, they inherit the host firewall. Eventually I gave up and fixed SpreadServe to run its RealTimeWebServer on port 80, and suddenly it worked!

Stuff that worked

There were three key points to getting SpreadServe going in Windows Containers...

- Two step process to build container image due to installer issues
- Web server inside the container should be on port 80 internally
- Create a one line launch script that sets up environment variables

The two step image building process is covered in detail above. I'll say a little more about the other two points here. Firstly the launch script. I mentioned above that I had to strip out the installer script code that created Registry entries for the two environment variables that SpreadServe needs: `SSROOT` and `SSROOTX`. To enable single line launch from docker I created a cmd script - `dbaseweb.cmd` - that sets the environment variables and launches SpreadServe. This enables me to launch SpreadServe instances like so:

```
docker run -p 80:80 ss031c c:\spreadserve\ss0.3.1\sh\dbaseweb.cmd
docker run -p 81:80 ss031c c:\spreadserve\ss0.3.1\sh\dbaseweb.cmd
```

To enable those launch lines to work the container host needs an Azure Endpoint defined for port 80, port 81 and any external port that needs mapping to a container. There also needs to be a firewall rule opening the port like so:

```
New-NetFirewallRule -Name "TCP81" -DisplayName "HTTP on TCP/81" -Protocol tcp -
↪LocalPort 81
```

Next steps

- SpreadServe's RealTimeWebServer authenticates against Windows user IDs and groups. How will that work in a container. Will groups and users be inherited from the host VM?
- Port mapping. I may want to run 1000 containers on the same host. Do I just write a single script to open 80 -> 1080, and have my container launch server use and reuse them as necessary?

CHAPTER 8

Indices and tables

- genindex
- modindex
- search