



MatrixSSL

Getting Started

TABLE OF CONTENTS

1	OVERVIEW.....	3
1.1	Who is this Document For?.....	3
2	COMPILING AND TESTING MATRIXSSL	4
2.1	POSIX Platforms using Makefiles	4
2.1.1	Preparation	4
2.1.2	Building the Source.....	4
2.1.3	SSL Self-Test Application	4
2.1.4	Sockets-Based Client and Server Applications.....	5
2.1.5	Debug Builds vs. Release Builds	6
2.2	WIN32 Platforms using Visual Studio Projects	6
2.2.1	Preparation	6
2.2.2	Building the Source.....	7
2.2.3	Self-Test Application	8
2.2.4	Sockets-Based Client and Server Applications.....	8
2.2.5	Debug Builds vs. Release Builds	11
3	MAC OS X PLATFORMS USING XCODE PROJECTS.....	12

1 OVERVIEW

This Getting Started Guide explains how to quickly compile and test the MatrixSSL package on supported reference platforms. This guide also contains instructions on building and running the client and server applications provided in the package.

1.1 Who is this Document For?

- Software developers working on a supported platform that want to create a development environment for integrating MatrixSSL security into a custom application
- Software developers who want to port MatrixSSL to a new platform
- Anyone wanting to learn more about MatrixSSL

2 COMPILING AND TESTING MATRIXSSL

2.1 POSIX Platforms using Makefiles

The POSIX classification in MatrixSSL encompasses support for several operating system platforms including Mac OSX 10.9.x and most UNIX/LINUX varieties. This is the default platform for the Makefile system that is provided in the package and should be the first build option if you are unsure of your platform configuration.

2.1.1 Preparation

The development platform must have the following tools installed:

- The *tar* archiver for unzipping the package (or other decompression utility supporting *.tgz* files)
- A C source code compiler and linker (GCC is the default in the provided Makefile system)
- The *make* tool

2.1.2 Building the Source

1. From the command prompt, unpack the zipped tar image.

```
$ tar -xzf matrixssl-3-6-2-open.tgz
```

2. Change directory to the root of the package and build the MatrixSSL library

```
$ cd matrixssl-3-6-2-open
$ make
```

3. Confirm there were no compile errors and that the MatrixSSL libraries have been built. A successful build will result in 6 binaries. There should be a shared and static library for each of the three modules of *./core/libcore*, *./crypto/libcrypto*, and *./matrixssl/libmatrixssl*. Applications link and interface with these libraries through the MatrixSSL public API set, which is documented in the [MatrixSSL API](#) reference, included in the distribution.

2.1.3 SSL Self-Test Application

Source code for a self-test application to exercise the SSL handshake and data exchange functionality of the MatrixSSL library is provided with the package. The following optional steps will enable the developer to build and run the test application to confirm the SSL protocol is fully functional.

1. Having successfully built the static library from the Building the source steps above, change directories to the test folder where the *sslTest.c* source is located and compile the application

```
$ cd matrixssl/test
$ make
```

2. Run the *sslTest* application from the command line. This sample output shows a successful run of the test using the default configuration of the open source package.

```
Testing TLS_RSA_WITH_AES_256_CBC_SHA suite
```

```

Standard handshake test
    PASSED: Standard handshake
Re-handshake test (client-initiated)
    PASSED: Re-handshake
Resumed handshake test (new connection)
    PASSED: Resumed handshake
Re-handshake test (server initiated)
    PASSED: Re-handshake
Resumed Re-handshake test (client initiated)
    PASSED: Resumed Re-handshake
Resumed Re-handshake test (server initiated)
    PASSED: Resumed Re-handshake
Change cert callback Re-handshake test
    PASSED: Upgrade cert callback Re-handshake
Change keys Re-handshake test
    PASSED: Upgrade keys Re-handshake
Change cipher suite Re-handshake test
    PASSED: Change cipher suite Re-handshake
Testing TLS_RSA_WITH_AES_128_CBC_SHA suite
Standard handshake test
    PASSED: Standard handshake
Re-handshake test (client-initiated)
    PASSED: Re-handshake
Resumed handshake test (new connection)
    PASSED: Resumed handshake
Re-handshake test (server initiated)
    PASSED: Re-handshake
Resumed Re-handshake test (client initiated)
    PASSED: Resumed Re-handshake
Resumed Re-handshake test (server initiated)
    PASSED: Resumed Re-handshake
Change cert callback Re-handshake test
    PASSED: Upgrade cert callback Re-handshake
Change keys Re-handshake test
    PASSED: Upgrade keys Re-handshake
Change cipher suite Re-handshake test
    PASSED: Change cipher suite Re-handshake

```

2.1.4 Sockets-Based Client and Server Applications

Source code for TCP/IP sockets-based client and server applications are provided with the MatrixSSL package. The following optional steps will enable the developer to build and run the applications to confirm the development platform is configured for MatrixSSL integration.

1. Having successfully built the library from the **Building the Source** steps above, change directories to the *apps* folder where the *client.c* and *server.c* source files are located and compile the applications.

```

$ cd apps
$ make

```

2. Run the *server* application from the command line

```

$ ./server
Listening on port 4433

```

3. In a second shell environment, run the client application using the Bash shell script and verify two connections were made to the running server. Client trace in the successful case:

```
$ ./runClient.sh
client https://127.0.0.1:4433/ new:1 resumed:1 cipher count:1
version:3.3
=== 1 new connections ===
Validated cert for: Sample Matrix RSA-1024 Certificate.
SEND: [GET / HTTP/1.0
User-Agent: MatrixSSL/3.4.3-COMM
Accept: */*
Content-Length: 0

]
RECV PARSED: [HTTP/1.0 200 OK]
RECV PARSED: [Server: MatrixSSL/3.4.3-COMM]
RECV PARSED: [Pragma: no-cache]
RECV PARSED: [Cache-Control: no-cache]
RECV PARSED: [Content-type: text/plain]
RECV PARSED: [Content-length: 9]
RECV COMPLETE HTTP MESSAGE
N
146 bytes received
0 msec (0 avg msec/conn SSL handshake overhead)
0 msec (0 avg msec/conn SSL data overhead)
=== 1 resumed connections ===
SEND: [GET / HTTP/1.0
User-Agent: MatrixSSL/3.4.3-COMM
Accept: */*
Content-Length: 0

]
RECV PARSED: [HTTP/1.0 200 OK]
RECV PARSED: [Server: MatrixSSL/3.4.3-COMM]
RECV PARSED: [Pragma: no-cache]
RECV PARSED: [Cache-Control: no-cache]
RECV PARSED: [Content-type: text/plain]
RECV PARSED: [Content-length: 9]
RECV COMPLETE HTTP MESSAGE
R
146 bytes received
0 msec (0 avg msec/conn SSL handshake overhead)
0 msec (0 avg msec/conn SSL data overhead)
```

2.1.5 Debug Builds vs. Release Builds

The default compiler options in the *Makefile* build system use the `-O2` or `-O3` optimization flag to create a release quality MatrixSSL library. If you wish to create a debug version of the library (and applications) edit the `common.mk` file at the top of the package directory and change the `BUILD` macro to `debug` rather than the `release` default.

2.2 WIN32 Platforms using Visual Studio Projects

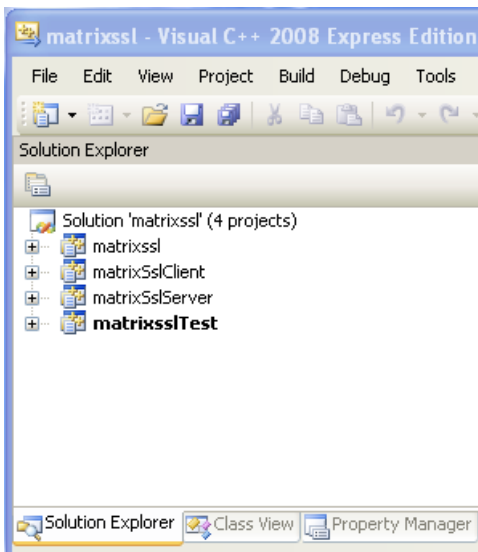
2.2.1 Preparation

The development platform must have the following tools installed:

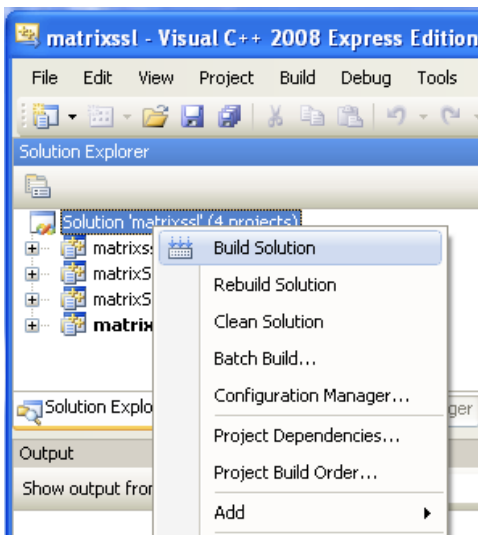
- Microsoft Visual C++ 2010 Express Edition

2.2.2 Building the Source

1. Unpack the product image to the directory of your choosing.
2. Open the *matrixssl.sln* file at the top level of the directory structure by either double clicking the file or choosing it through the *File -> Open -> Project/Solution* menu option of Visual C++. The result should be a solution with four projects.



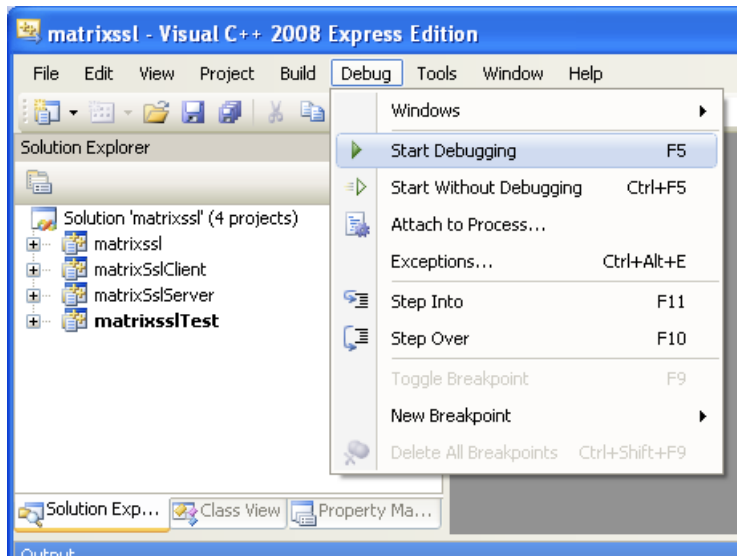
3. Build the package by right clicking the top-level solution and selecting Build Solution.



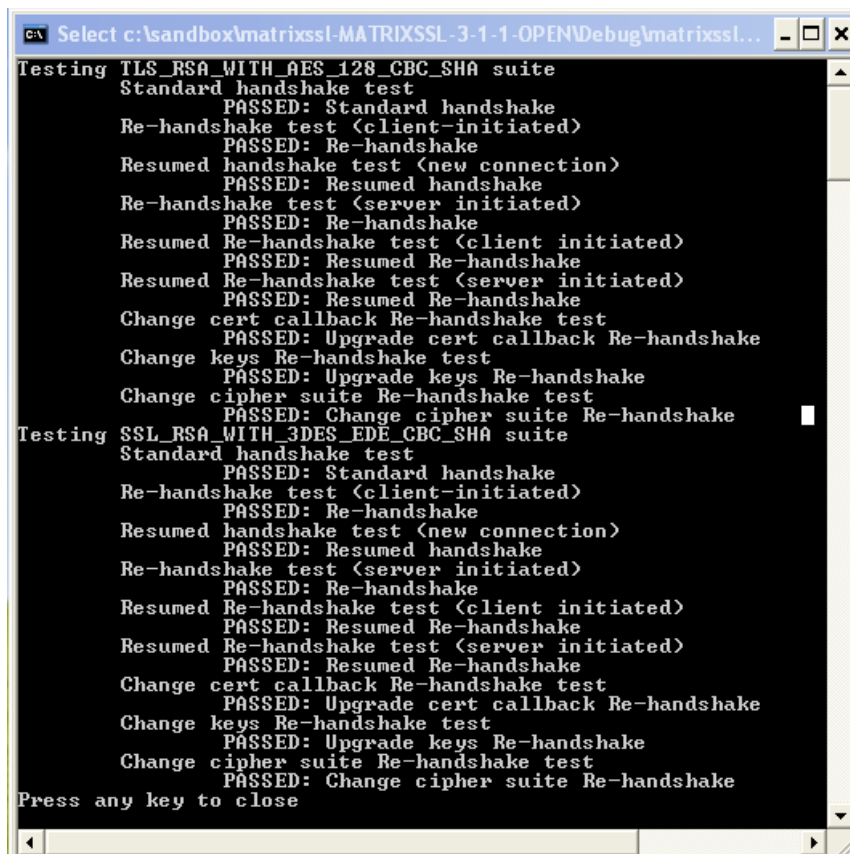
4. Confirm there were no compile errors and that the MatrixSSL library and applications have been built. A successful build will result in a *Debug* (or *Release*) directory being created at the top-level directory with the object files, libraries, and executables. The *matrixssl.dll*, *matrixSslClient.exe*, *matrixSslServer.exe*, and *matrixssl/Test.exe* files are the final outputs to look for.

2.2.3 Self-Test Application

The test application will have been built using the above steps. To run the test application, ensure the *matrixssl/Test* project appears in bold in the Solution Explorer to indicate it is the StartUp project. Select *Debug -> Start Debugging* from the menu to start the application.



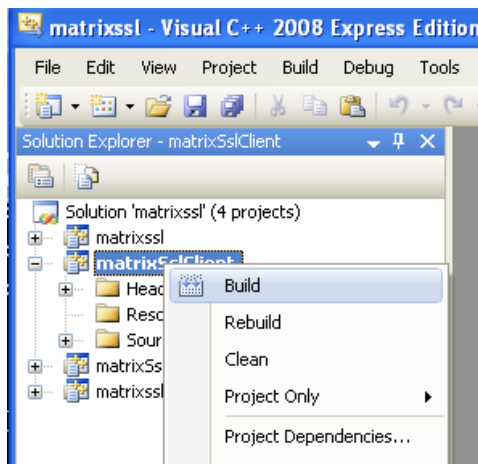
The console output should look like this.



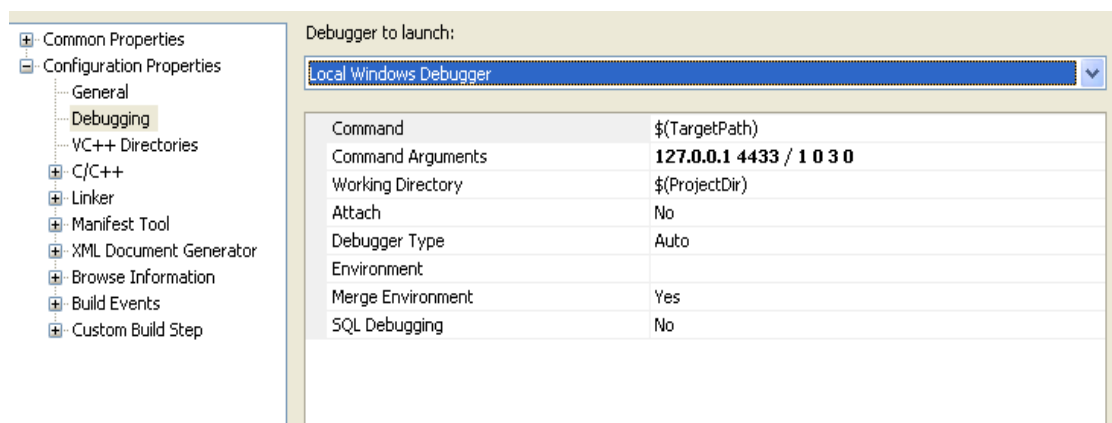
2.2.4 Sockets-Based Client and Server Applications

Source code for TCP/IP sockets-based client and server applications are provided with the MatrixSSL package. The client and server applications will have been built when the Solution was built in the above steps.

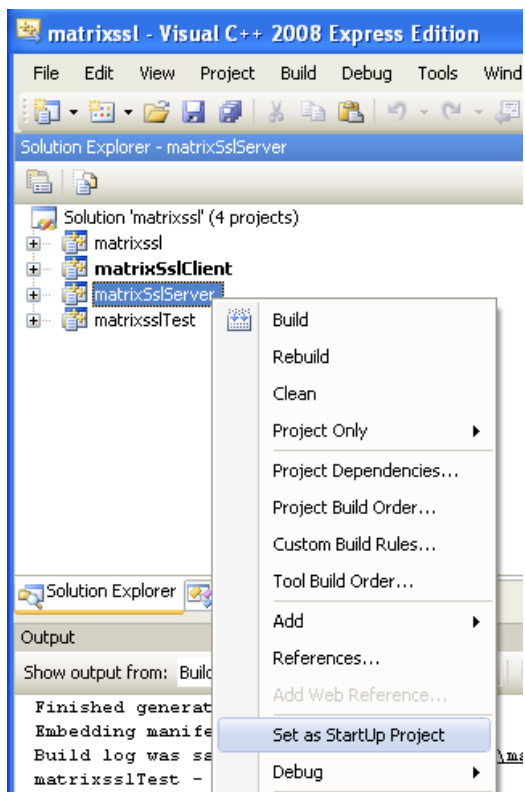
To individually recompile a project, simply right click the project name and choose Build from the drop down list.



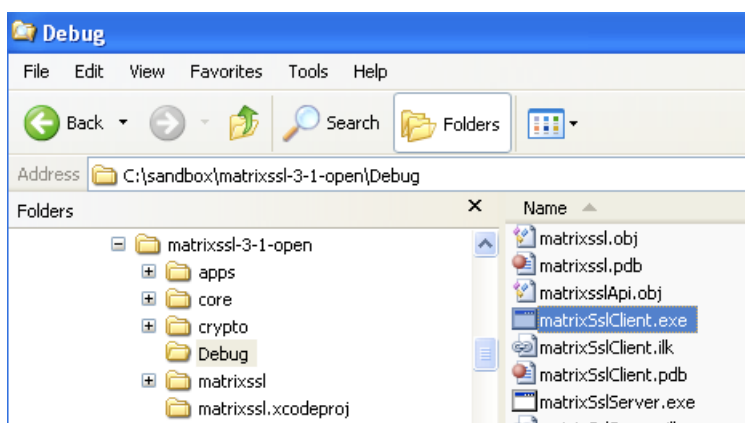
The client application must be given command line options that nominate the server IP address, port, session counts, protocol version, and cipher suites. To set the command line options, select Debugging from the Configuration Properties section of the Property Pages and enter the line in the Command Arguments. The default setting would be "127.0.0.1 4433 / 1 0 3 0".



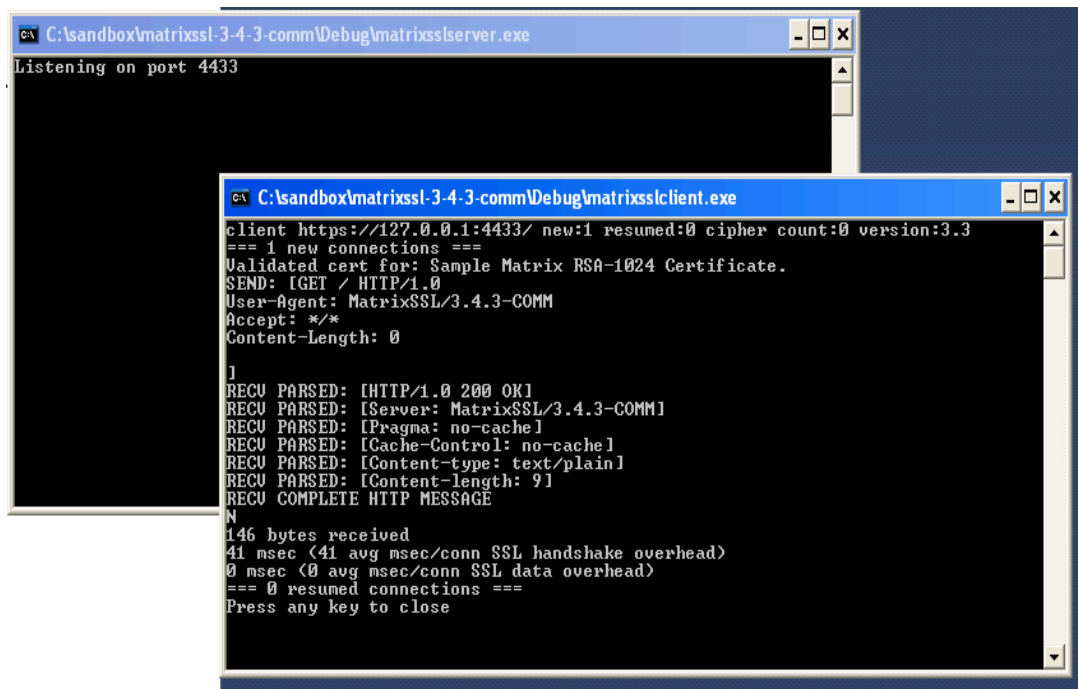
To run the client or server application in the Visual C++ environment, ensure the target project appears in bold in the Solution Explorer to indicate it is the StartUp project. Select *Debug -> Start Debugging* from the menu to start the application.



Only a single project can be executed from within the Visual C++ environment at a time so when testing the server and client applications against each other, at least one of them must be manually run by double clicking the executable file. The server application must be started before the client.

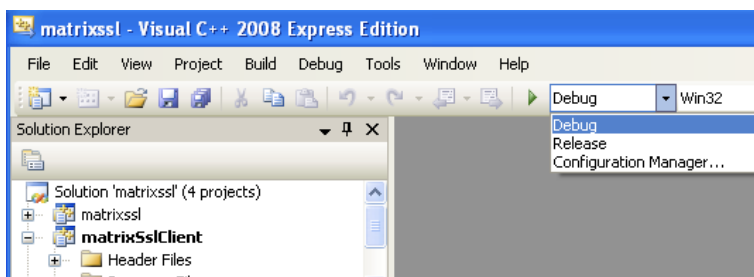


A successful test will result in console output similar to this.



2.2.5 Debug Builds vs. Release Builds

The default build configuration for the provided Visual Studio projects is set to Debug. If you wish to create release versions of the library or applications, simply select the Configuration pull-down list and choose Release as shown below.



By default the output files will be created in a directory at the top level matching the name of the build (Debug or Release).

3 MAC OS X PLATFORMS USING XCODE PROJECTS

Xcode projects have been included for building the MatrixSSL library, the test application, and the client/server applications. The Xcode version most recently used to test the projects was 5.0.2. The version of Mac OS X most recently used to test the projects was 10.9.1

Xcode schemes for projects are saved to disk on a per-user basis so the client application scheme will need to be edited to pass the command line options to the executable. With the matrixClient project selected in the workspace, choose “Product – Scheme – Edit Scheme” to bring up the “Arguments Passed On Launch” window. Add an argument string of “127.0.0.1 4433 / 1 0 3 0” to connect to a localhost server using TLS 1.2.