Introduction

This document provides guidelines on building a basic framework of an XData server and a Sphinx server used for authentication and authorization.

In the example code provided, both servers have been implemented as windows services because this is considered to be the most logical setup for most production environments.

Disclaimer

This framework is provided "as is" and by no means intended as a complete framework for all situations. Each implementation of such a framework will have its own requirements and require specific adjustments which are entirely up to the user to implement.

Therefore, this document is to be considered as a "getting started guide" only. It can help new users new to XData and/or Sphinx to quickly get started building their own setup, whilst avoiding common pitfalls and misunderstandings encountered by the author.

All comments, suggestions and improvements are welcome and should be posted on the TMS Sphinx forum, in the post in which this document and associated code were posted.

1. Project group

The entire project group can be downloaded from GitHub at:

https://github.com/Softdrill-BV/XData_Sphinx_Services

The project group comprises the following projects

- **XData Server** (Windows Service Application)
- **Sphinx Server** (Windows Service Application)
- **Sphinx Manager** (VCL Desktop Application)
- **VCL Client** (VCL Desktop Application)

All projects are built with standard Delphi components (i.e. no third-party components). However, logging features are implemented with TMS Logging as it is assumed that users having Aurelius, XData and Sphinx will also have access to TMS Logging.

The entire project group is stored in a single directory. There are no hard-coded paths in the application (except file names), so it may be stored in any folder. The folder does include a sub-folder `..\Win32` in which some further files are stored:

- **sqlite3.dll**[*]. All databases used (Sphinx and XData) are SQLite.
- **libeay32.dll** & **ssleay32.dll**[*]. OpenSSL files, required for https communications.
- **GENERATE_KEYS.bat**. A script to generate the private and public encryption key files.
- Some batch files to simplify (un)installing and starting (stopping) the XData and Sphinx Services. These scripts must be run as Administrator.

[*] *These files are not included in the repository. It is up to the user to source them and place them in the applications folder.*

All 32-bit build configurations are configured to output the final *.exe in the `..\Win32` sub-directory. All dcu's are output to `..\Win32\dcu`.

The XData server is defined with a default base URL (http://+:2001/example/sphinx). If not yet done, the base URL will be registered in code so that use of the external TMSHttpConfig tool is not required.

And finally two sub-directories (`..\Win32\img` and `..\Win32\css`) are included that contain some files to customize the Sphinx login page etc.

To run and test the framework, the following steps must be performed:

1. Clone the repository a directory of choice.

2. Add the required DLL's to the `\Win32` sub-directory
3. Generate the key files for asymmetric encryption by running the `GENERATE_KEYS` batch file in the `\Win32` sub-directory.
4. Build entire project group.
5. Install and start the services by running the `Services_RegisterStart` batch file with administrative rights.
6. Run the Sphinx Manager application to add a user (ref. section 1.3 on page 5).
7. Run the VCL Client application and log in as the newly created user.

Note

For steps 6 and 7 to work, the Sphinx server and Manager application must first be extended with email capabilities or some other form of issuing passwords and confirmation tokens.

1.1. XData Server

The XData server is a very simple example exposing only three entities, defined in unit `Entities.Example.pas`. The most essential part of this implementation (in conjunction with the Sphinx server) is that access to the entities is controlled by `EntityAuthorizeScopes` attributes and a 'scope' claim issued by the Sphinx server in the JWT token (see Sphinx Server notes on page 4).

The server uses (or creates) a SQLite database in the application folder:

`..\Win32\XDataService.db`.

The XData server was initially created through the TMS provided template (File > New > Other > TMS XData). This creates the server as a desktop application with a module for the XData server. This module, contained in `Module.XDataServer.pas`, was simply added to (i.e. copied) a new Service application after which the desktop application was discarded.

The XDataServer component has the JWT middleware added at design-time (right-click to access list). The middleware implements the `XDataServerJWT.OnGetSecretEx` event handler in which the JWT token is validated against the Key ID and public key (`..\Win32\my-public.key`). The Key ID is defined as a constant (arbitrary value) and **must** be the same in the XData and Sphinx servers. Please refer to Sphinx server on page 5 for more details.

1.2. Sphinx Server

Similar to the XData server, the Sphinx server was first created through the TMS provided template. However, this creates a desktop application with the entire Sphinx server (i.e. all components) in a form. To add this functionality to a Windows service application, a new data module was created (`Module.SphinxServer.pas`) and all components and code were copied into this module.

The server uses (or creates) a SQLite database in the application folder:

`..\Win32\SphinxService.db`.

Note

The Sphinx server implements some email functionalities through a class `TSphinxMailer`. This class is not included and all related code is commented out. The user of this framework is expected to implement his/her own emailing functionality which is required to notify users of account details and confirmation tokens.

The Sphinx server implements an extended user entity (`TSphinxUserEx`) that adds the property `Access_Level`. The user type (discriminator value for Aurelius inheritance) is 'sphinx_ex'. In all applications, which is defined by the statement:

```
SphinxConfig.UserClass := TSphinxUserEx;
```

Four access levels are defined for TSphinxUserEx: ***reader***, ***editor***, ***poweruser*** & ***admin***.

Access to the XData entities, through `EntityAuthorizeScopes` attributes in the entities, is restricted as follows:

Entity	User Level	Access
TInvoice	reader	Read
	editor	Read, Write
	poweruser	Read, Write
	admin	Read, Write
TInvoiceItem	reader	Read
	editor	Read, Write
	poweruser	Read, Write
	admin	Read, Write
TSecret	reader	None
	editor	None
	poweruser	Read
	admin	Read, Write

Note that `access_level` is cumulative. the `EntityAuthorizeScopes` are defined as such that a user will also have the access rights of the lower access levels. I.e. an Editor has write access but also (implied) read access.

All set up is done in `DataModuleCreate`. Most of these settings (e.g. Client definition) can be done design-time from the IDE but have been included here to demonstrate what is required.

Note that `ReadConfigFromIni` in the Sphinx server implementation is a public method and called from `Module.SphinxService` when starting the Windows service It configures the server from an ini file on each start up (see section 1.3 Sphinx Manager below). This can probably done from the `OnStart` event handler for the `SparkleHttpSysDispatcher` but in the current implementation this yields an exception. Solving this issue is on the to-do list (ref. Section 2 on page 7) and the current implementation is considered to be an acceptable work-around.

Two event handlers in the Sphinx server are important:

In the `SphinxConfig.OnConfigureToken` event handler we can add whatever is required to the token issued. In this case we're adding the 'scope' claim, based on the `Access_Level` of the user (`TSphinxUserEx`). In online examples, this is also where – for example – the Tenant ID is added to the token.

Asymmetric encryption of the token is implemented in `SphinxConfig.OnGetSigningData` where the JWT token `KeyId` is set and it is signed with the ***private*** key (`..\Win32\my-private.key`).

Important to note here that this code only adds the `KeyId` to an ***existing*** JWT token and signs it with the `Key` data. It does **not** generate a JWT token. The JWT token is automatically created by the Sphinx server ("under the hood") which may not be entirely clear from the documentation.

1.3. Sphinx Manager

The Sphinx Manager is a simple VCL desktop application that must be run from the same directory as the Sphinx server. The application has two functionalities:

- A means to configure the Sphinx server. All changes made on the first tab will influence the server behaviour and must be save manually (to `..\Win32\ SphinxService.ini`). After saving the configuration, the user is notified that the server must be (re)started for the changes to take effect. The batch files included (i.e. `Service_Stop.bat` and `Service_Start.bat`) can be used to simplify this task.
- A simplified user manager to add, edit or remove users from the database.

Most code will be self-explanatory but the use of `TwoFactorRequired` and `TwoFactorEnabled` for users requires some attention. To make 2FA *mandatory* for a user, it is required to set `TwoFactorRequired := true`. If `TwoFactorEnabled` is `false` or has no value (= `null`), the user will be presented with a QR code to set up an authenticator app on the first login. Both Microsoft Authenticator and Google Authenticator can be used for this purpose.

Setting `TwoFactorEnabled` to `true` will **not** work as the user will then **not** be presented with a QR code on the first log in. As such he/she cannot setup 2FA and will never able to fully log in.

So, `TwoFactorEnabled` is not about 2FA being *functional*. It is about the set up thereof having been *completed* by the user. Setting `TwoFactorEnabled` to false will result in the user having to set up the 2FA again. For security reason, the authenticator key is then internally reset as well.

Note

Like the Sphinx server, the Sphinx Manager implements some email functionalities that has been commented out. It is up to the user of this framework to implement his/her own emailing functionality.

1.4. VCL Client

A very simple XData client desktop application to test the framework and demonstrate the implementation of `TSphinxUserEx.Access_Level` and associated claim 'scope' in the authorization token.

Note that this application may be run from any location as long as it has access to the computer running Sphinx and XData on the designated port (default 2001). When run on another computer than the services, firewall adjustments may be required.

Noteworthy in this application is the 'Get secrets' button. This will retrieve a list of `TSecret` entities (as defined in `Entities.Example`) but *only* if the authorized user has `Access_Level` **poweruser** or **admin**. Only access level **admin** may modify or delete entities (see page 4 for further details).

The `PrepareDatasets` procedure applies settings to all Aurelius Datasets to make it all functional. Especially the `TAureliusDataset.RefreshObjectOnCancel := true` statement. If not set and the user has to cancel edits (e.g. because of insufficient access rights), the edit is cancelled but the entity will still contain the new (unsaved) values and is inconsistent with the database.

2. To do

There are certainly things that can be improved on or added to this framework.

- Fix the implementation of the `SparkleHttpSysDispatcher.OnStart` event handler in the Sphinx server. This way, the `ReadConfigFromIni` call can be removed from the service start and made private or protected again (i.e. improve encapsulation).
- Identify where (and how) to best implement logging of user logins, in Sphinx (token issued to whom and when) and XData (who logged in and when).
- Find a good way to handle access token life time (now set to 3600 seconds or 1 hour) and expiry. Now the user may be inactive for some time and when resuming work, the token may have expired and pending edits cannot be saved (401 – token expired).