LinearBase

How-To Host a Databank Program

Release 0.1

Included

- LinearBase.Host.zip
- LinearBase.Server.nuget available from https://nuget.org
- LinearBase.Client.nuget available from https://nuget.org
- Docker image linearbase/lineardb-amd64:latest available from https://hub.docker.com

Required Artifacts

- Databank files
- Databank Plain Old CLR Object (POCO) code
- LinearBase.Host test project
- Profiles supporting Standalone, Centralised, Decentralized, and Distributed deployments
- Initializers for use with LinearBase.Host.exe

Summary

Once created using the Transform or Model processes, the next step is to Host the Databank files as in-memory Programs across devices and platforms such as smart-phones, tablets, laptops, desktops, and servers.

There are two ways to achieve this

1. The developer includes the necessary Profile initiation instructions within mobile and business Apps using either LinearBase.Server.nuget or LinearBase.Client.nuget packages making data processing an integral aspect of the resulting deployment.
2. System Admins use the auto-generated initialization scripts to create micro-service Program instances using the free-to-download Docker-on-Linux or Linearbase.Host.exe programs.

Profiles control everything in LinearBase describing Programs to load, Program Locations, Services to load, and Service Locations.

In this article we'll cover how to compile and run the Example.Host project, auto-generated for each successful Transform or Model process, and use auto-generated launch options to create a **Standalone** Primary Active Realm Service (PARS) instance. Once we have an active PARS instance we'll run the Example.Host program as a **Distributed** Client communicating with the PARS instance.

In this release Dynamic Client Port assignment is disabled, instead the initial PARS port value is reused. If two or more instances are invoked on the same device, that is the same IP address, it results in a warning saying Client not listening on port x as port already in use.

A future release will introduce the Profile Manager Portal to create and maintain Profiles including Dynamic Client Port assignment. It does not affect this demonstration.

Download the LinearBase.Host.exe Program

If you have already downloaded and extracted the LinearBase.Host.exe prom you can skip this section.

Host Program

Navigate to https://linearbase.co/#download

Click or tap the 'Download Host' button. This will download a 31Mb compressed file called 'Host.zip to your device's download folder.

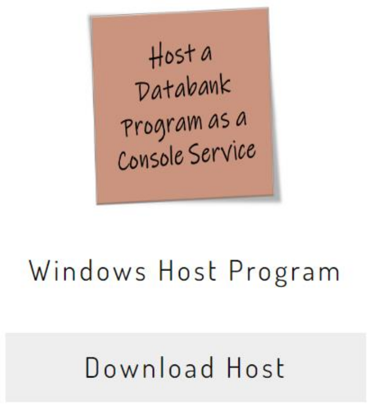


Figure 1 Download Widows Host Program

Extract the compressed files to the folder, for example 'E:\LinearBase', by selecting Host.zip and choosing the 'Extract All…' option from the context menu.

Unless you change the default options this will create a new folder E:\LinearBase\Host.

There are three files in the zip archive, once decompressed the file we're interested in is LinearBase.Host.exe. The program requires the other two support files to run.

The LinearBase.Host.exe program is initiated using launchOptions.json configuration file or by command line arguments. The transformation process auto-generates these settings for both options and writes them to the relevant Realm folder upon completion.

Inspect Source Files

Assuming successful transformation, navigate to the Realm path, for example E:\LinearBase\Realms\Adventure.

In this example we'll use the AdventureWorks2017.Sales Program created during the article How-To Digitally Transform an Existing Database into a Databank Program and Code.

Auto-generated identifiers are unique so will not match those used in the example, substitute your values where necessary.

Navigate to and open one of the Transformed or Modelled Schema folders.

If you haven't done so already select Expand.ps1, right click and choose 'Run with PowerShell'. This generates another folder named ExampleHostConsole containing the zip file contents.

Expand the …\ExampleHostConsole folder to inspect unzipped files…

Example.Host Project

Open Example.Host.csproj in Visual Studio, the current version is 2022, or VS Code.

Example.Host Program File

Open file Program.cs.

This is a Generic Host implementation demonstrating the AddServiceExtension method. It highlights how one line of code, AddSales() in the example, replaces all data connections, data contexts, and scaffolding. The 'Publish' method in the extension class invisibly populates the application with data from .data file created during the transformation process along with any subsequent insert, update, or delete transaction actions.

To the developer, all data specific functionality is inline and part of the application logic.

Add further Add…() statements to register any mix of Programs (in-memory), Locations (address of), and Services (logic).

AddSalesExtension File

Open file AddSalesExtension.cs. In the Create & Publish region there are two Boot instructions and three Publish; all bar one are commented out.

As Sales was the first Transformation in the example article instruction B is uncommented. If the Transformation or Modelling process appended a Program instruction D will be uncommented.

To learn about Boot and Publish Profiles read the use-case article The 16 Configurable Topologies of Cloud*LESS* Data Processing.

In summary:

a. Profile type A – Boot. Start a Primary Active Realm Service (PARS) instance from the file system. This is the controller in the default '**Standalone**' state, that is there is only system information loaded. The instance is listening on the configured port. Clients initialize communication by initiating Profile C described below on application load.
b. Profile type B – Publish. Start a Primary Active Realm Service (PARS) instance from the file system and Publish a Databank Program into memory on the same instance. The instance is listening for Client requests on the configured port. This is the controller in a '**Centralized**' state, that is there is system and Program information loaded. Clients initialize communication by initiating Profile E described below on application load. Alternatively, Clients can ingest the same Program using Profile D described below. The Client acts as processor and/or server thereby distributing processing to any device or platform simultaneously.
c. Profile type C – Boot. Join a Client application to a Realm but don't load any Programs. This allows the Client to listen for run-time instructions to load Realm specific Programs on-demand using Profiles D or E described below.
d. Profile type D – Publish. Join a Client application to a Realm, if it isn't joined already, and load one or more Programs, Program Locations, Services or Service Locations. Depending on the extension used the instance optionally acts as a Server listening for other Client requests on the configured port. This is the Client in '**Distributed**' state when inside the firewall and '**Decentralized**' when outside.
e. Profile type E – Publish. Join a Client application to a Realm, if it isn't joined already, and load one or more Program Locations, or Services Locations. Here the Client doesn't perform any data processing and is considered an '**Agent**'.

The benefit of Profiles is developers only programme once. The logic is constant, Profiles dictate where the processing occurs across the enterprise. There are no data contexts or connection strings; all communication is now managed by the LinearBase programming language extensions Linearbase.Server and LinearBase.Client.

ExampleHostedServce File

Open file ExampleHostedService.cs.

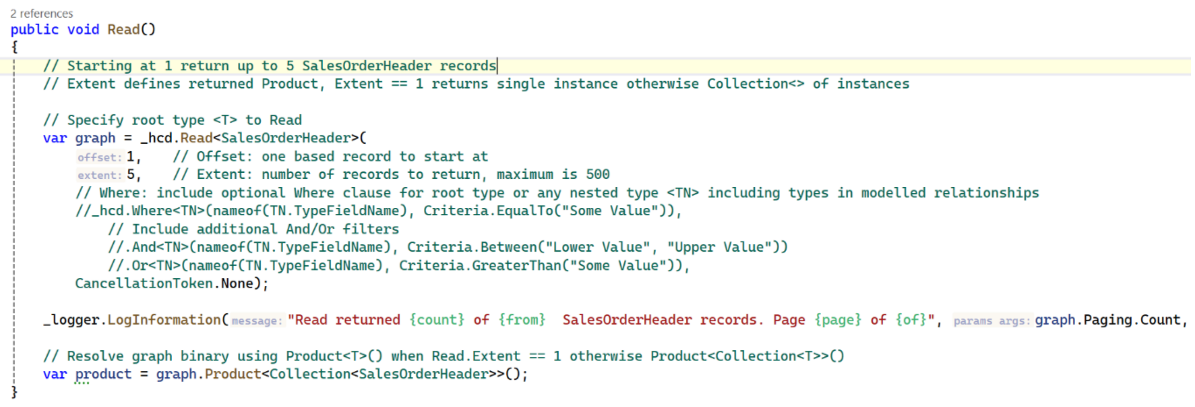
The demonstration uses a Hosted Service. According to Microsoft a hosted service is a class with background task logic that implements the IHostedService interface. We use this to handle the Start and Stop tasks associated with the program which in turn invokes the Read() and Write() methods on the Published Program.

ExampleReadWrite File

Open file ExampleReadWrite.cs.

Read Section

In the IExampleReadWrite implementation region examine the Read() method…



```
2 references
public void Read()
{
    // Starting at 1 return up to 5 SalesOrderHeader records
    // Extent defines returned Product, Extent == 1 returns single instance otherwise Collection<> of instances

    // Specify root type <T> to Read
    var graph = _hcd.Read<SalesOrderHeader>(
        offset: 1,    // Offset: one based record to start at
        extent: 5,    // Extent: number of records to return, maximum is 500
        // Where: include optional Where clause for root type or any nested type <TN> including types in modelled relationships
        //_hcd.Where<TN>(nameof(TN.TypeFieldName), Criteria.EqualTo("Some Value")),
            // Include additional And/Or filters
            //.And<TN>(nameof(TN.TypeFieldName), Criteria.Between("Lower Value", "Upper Value"))
            //.Or<TN>(nameof(TN.TypeFieldName), Criteria.GreaterThan("Some Value")),
        CancellationToken.None);

    _logger.LogInformation(message: "Read returned {count} of {from}  SalesOrderHeader records. Page {page} of {of}", params args: graph.Paging.Count,

    // Resolve graph binary using Product<T>() when Read.Extent == 1 otherwise Product<Collection<T>>()
    var product = graph.Product<Collection<SalesOrderHeader>>();
}
```

Figure 2 - Read Method

During the example Transformation process we identified SalesOrderHeader as the Principal. The auto-generated code will use whatever you chose in your Transform or Model process.

The Read<*root-type*>() method accepts any type in the Classes folder. The only caveat is types entered as Where clause filters must be a nested type, that is a type equal or lower in the hierarchical object graph. It cannot be a parent of the type or unrelated otherwise an incident exception is raised.

Decide which type or types will feature in the Where clause, if any.

Optionally include one or more Where clause filters for the root type, the one specified in Read<*root-type*>(), or any nested type <TN> including types in modelled relationships. Relationship Modelling is a feature of the Domain Model Portal and available in a future release.

Each Where clause accepts the type and a field name of the type upon which to filter. And + Or Extension methods provide Where clauses chaining.

The Read<*root-type*>() method has four parameters, three required and one optional:

- Offset – one based record to start from within the result set, required
- Extent - number of records to return, maximum is 500, required
    - 1 signifies return the first *root-type* instance, no array
    - > 1, return an Collection<*root-type*> of instances
    - < 0, return an Collection<*root-type*> of instances in reverse order
- Where – optional, in the form of a single Where followed by one or more And + Or
    - _hcd.Where<TN>(nameof(TN.TypeFieldName), Criteria.EqualTo("Some Value")),
    - .And<TN>(nameof(TN.TypeFieldName), Criteria.Between("Lower Value", "Upper Value"))
    - .Or<TN>(nameof(TN.TypeFieldName), Criteria.GreaterThan("Some Value"))
    - Etc.
- CancellationToken - required

In this example we'll use Offset 1, Extent 20 and Where clause on SalesOrderHeader.CustomerID knowing SalesOrderHeader with CustomerID 11019 has 17 records:

_hcd.Where<SalesOrderHeader>(nameof(SalesOrderHeader.CustomerID), Criteria.EqualTo("11019")),

Read returns an instance of a Graph object containing the resulting object graph binary and paging information.

Paging has four properties

- Page – the returned page
- Of – the total number of possible pages for the given Read criteria
- Count – the number of records in this page, aligned to Extent, current limit is 500
- From – the total number of records for the given Read criteria

To deserialize the Graph.Binary into an instance or instances of the principal object call graph.Product().

For example, when extent = 1 pass Product() the principal type only, i.e.,

var product = graph.Product<SalesOrderHeader>();

For any other extent pass Product() a collection, i.e.,

var product = graph.Product<Collection<SalesOrderHeader>>();

Place an optional breakpoint on the 'var product' line to inspect the returned object graph.

Write Section

There are two approaches to creating a Databank and code, data-first and code-first.

The article How-To Digitally Transform an Existing Database into a Databank Program and Code explains data-first. When using this method associations between objects use the Entity Relational approach, that is, the parent object has the key and the child holds a reference to the key, the foreign key.

The article How-To Digitally Model a New Databank Program and Code explains code-first. When using this method associations between objects use the Object Relational approach, that is, the child object has the key and the parent holds a reference to the key, the foreign key.

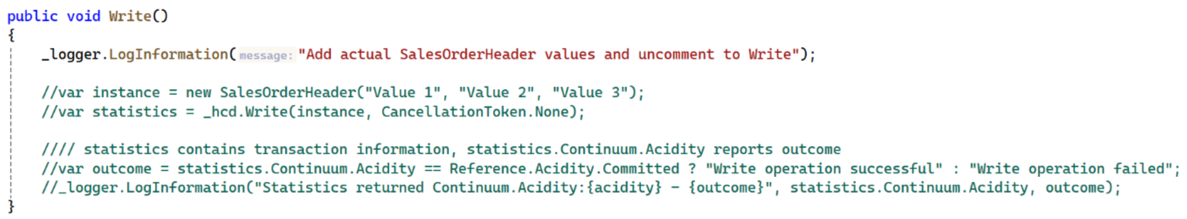
Typically, an automatic unique sequence generator or some other manual method manages key allocation, LinearBase is no different. As LinearBase is naturally distributed a random context unique identifier is used, referred to as an Id8. To accommodate this, primary and foreign key fields become read-only text fields regardless of their original data type.

Data-first or code-first dictates whether or not primary and foreign keys are visible in code. Data-first has them, code-first does not, they're implied by object references.

With the data-first approach you'll see primary and foreign key fields as read-only properties in classes and interfaces. When creating instances, the constructor controls how to populate fields depending on whether they're new or pre-existing, that is, created during a Read() request.

Regardless of data-first or code-first, the Read and Write process automatically manages and assigns primary and foreign key allocations. Do not populate these fields.

A single line of code controls the entire transactional insert and update process. Delete functionality omitted for this release.



```
public void Write()
{
    _logger.LogInformation(message: "Add actual SalesOrderHeader values and uncomment to Write");

    //var instance = new SalesOrderHeader("Value 1", "Value 2", "Value 3");
    //var statistics = _hcd.Write(instance, CancellationToken.None);

    //// statistics contains transaction information, statistics.Continuum.Acidity reports outcome
    //var outcome = statistics.Continuum.Acidity == Reference.Acidity.Committed ? "Write operation successful" : "Write operation failed";
    //_logger.LogInformation("Statistics returned Continuum.Acidity:{acidity} - {outcome}", statistics.Continuum.Acidity, outcome);
}
```

Figure 3 - Write Method

The Write method accepts an instance or collection of instances of any object hierarchy in the Classes folder. The process detects ANY changes within the supplied object graph and transactionally persists updates to the correct Databanks and Programs.

To save a change, first create a new object instance or read an existing instance by repeating the Read process above. Update one or more fields and call _hcd.Write(…) to distribute and persist the changes, it is that simple.

Decide upon the class to create. Modify the 'var instance = new …' signature passing the relevant values. Create an entire new object graph if the new instance is complex and contains references to other types.

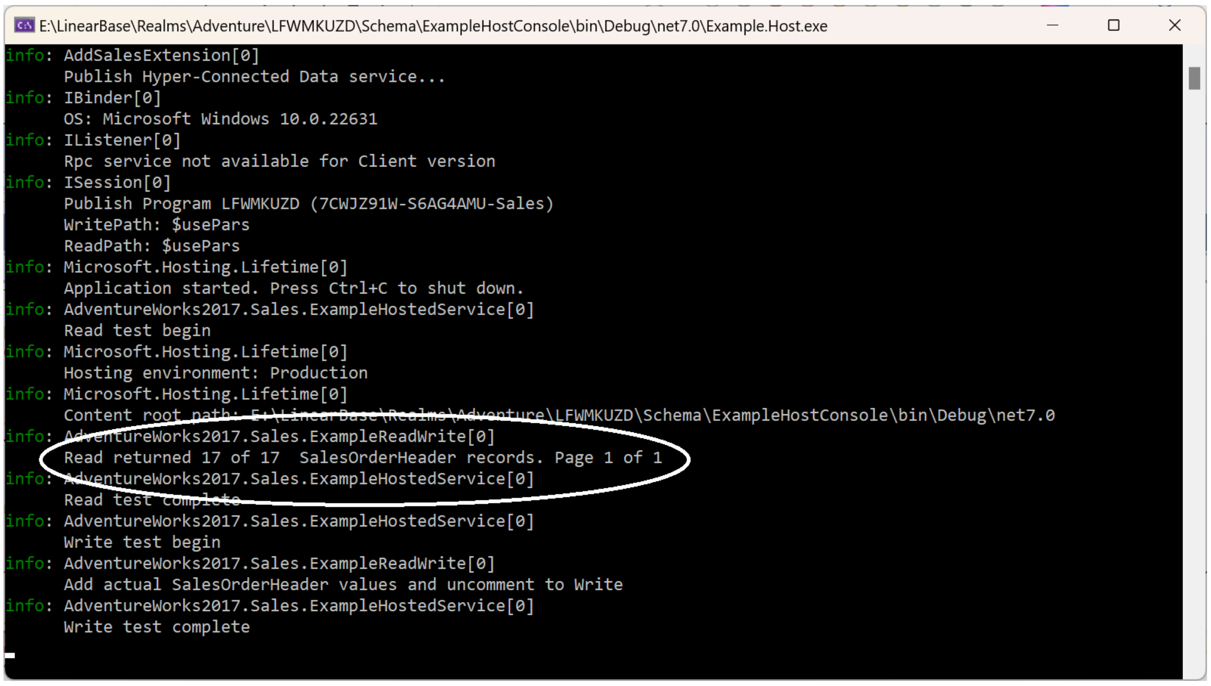
As mentioned earlier, identity fields become read-only during the transformation process; the write process manages their population and associations.

On completion, the Write(…) process returns a Statistics instance detailing the outcome of the operation. The primary notifier is the *statistics.Continuum.Acidity* value. On a successful operation the response is Reference.Acidity.Committed, responses other than this signify failure, such as a data consistency violation. In these situations, the RevisionId field contains a reference to the incident.

Test It

The application is a simple demonstration of how to host the service with integrated data processing within a program with basic run-once read and write examples.

Compile and run the application.



E:\LinearBase\Realms\Adventure\LFWMKUZD\Schema\ExampleHostConsole\bin\Debug\net7.0\Example.Host.exe

```
info: AddSalesExtension[0]
      Publish Hyper-Connected Data service...
info: IBinder[0]
      OS: Microsoft Windows 10.0.22631
info: IListener[0]
      Rpc service not available for Client version
info: ISession[0]
      Publish Program LFWMKUZD (7CWJZ91W-S6AG4AMU-Sales)
      WritePath: $usePars
      ReadPath: $usePars
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: AdventureWorks2017.Sales.ExampleHostedService[0]
      Read test begin
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: E:\LinearBase\Realms\Adventure\LFWMKUZD\Schema\ExampleHostConsole\bin\Debug\net7.0
info: AdventureWorks2017.Sales.ExampleReadWrite[0]
      Read returned 17 of 17  SalesOrderHeader records. Page 1 of 1
info: AdventureWorks2017.Sales.ExampleHostedService[0]
      Read test complete
info: AdventureWorks2017.Sales.ExampleHostedService[0]
      Write test begin
info: AdventureWorks2017.Sales.ExampleReadWrite[0]
      Add actual SalesOrderHeader values and uncomment to Write
info: AdventureWorks2017.Sales.ExampleHostedService[0]
      Write test complete
```

Figure 4 - Example Host Program Output

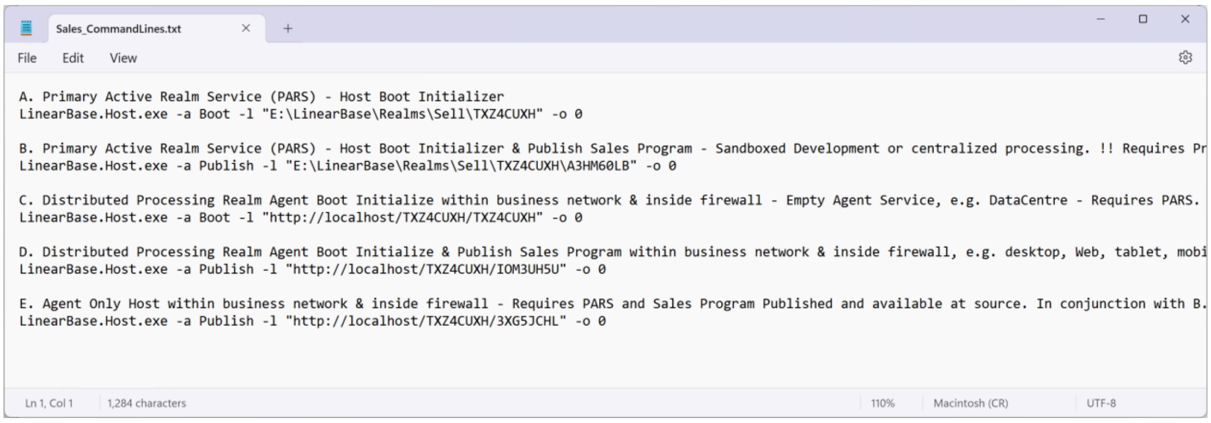
The output shows the Read request returned 17 results as expected in our test data example.

Initialization Scripts

Each successful deployment auto-generates four initialization scripts prefixed by the supplied Alias, in the example it’s Sales, saved to the ‘initializers’ folder under the Realm name folder.

Each file contains the five Profile scenarios discussed earlier covering Standalone, Centralized, Distributed, and Decentralized deployments.

Command Lines

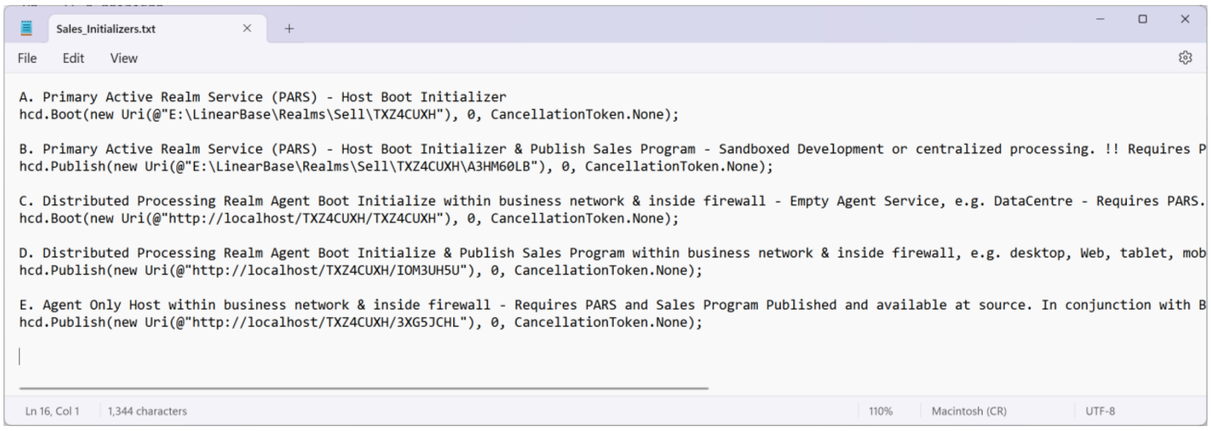


```
A. Primary Active Realm Service (PARS) - Host Boot Initializer
LinearBase.Host.exe -a Boot -l "E:\LinearBase\Realms\Sell\TXZ4CUXH" -o 0

B. Primary Active Realm Service (PARS) - Host Boot Initializer & Publish Sales Program - Sandboxed Development or centralized processing. !! Requires Pr
LinearBase.Host.exe -a Publish -l "E:\LinearBase\Realms\Sell\TXZ4CUXH\A3HM60LB" -o 0

C. Distributed Processing Realm Agent Boot Initialize within business network & inside firewall - Empty Agent Service, e.g. DataCentre - Requires PARS.
LinearBase.Host.exe -a Boot -l "http://localhost/TXZ4CUXH/TXZ4CUXH" -o 0

D. Distributed Processing Realm Agent Boot Initialize & Publish Sales Program within business network & inside firewall, e.g. desktop, Web, tablet, mobi
LinearBase.Host.exe -a Publish -l "http://localhost/TXZ4CUXH/IOM3UH5U" -o 0

E. Agent Only Host within business network & inside firewall - Requires PARS and Sales Program Published and available at source. In conjunction with B.
LinearBase.Host.exe -a Publish -l "http://localhost/TXZ4CUXH/3XG5JCHL" -o 0
```

Figure 5 - Sales Command Lines

Command lines is used in conjunction with LinearBase.Host.exe. Open a Command window and change directory to the one containing LinearBase.Host.exe. This executable is a wrapper around LinearBase.Server and will listen for Client requests on the supplied port. Change the port number of the relevant Profile instruction to between 2000 and 65535. Copy and paste the instruction into the command window and press return.

Initializers

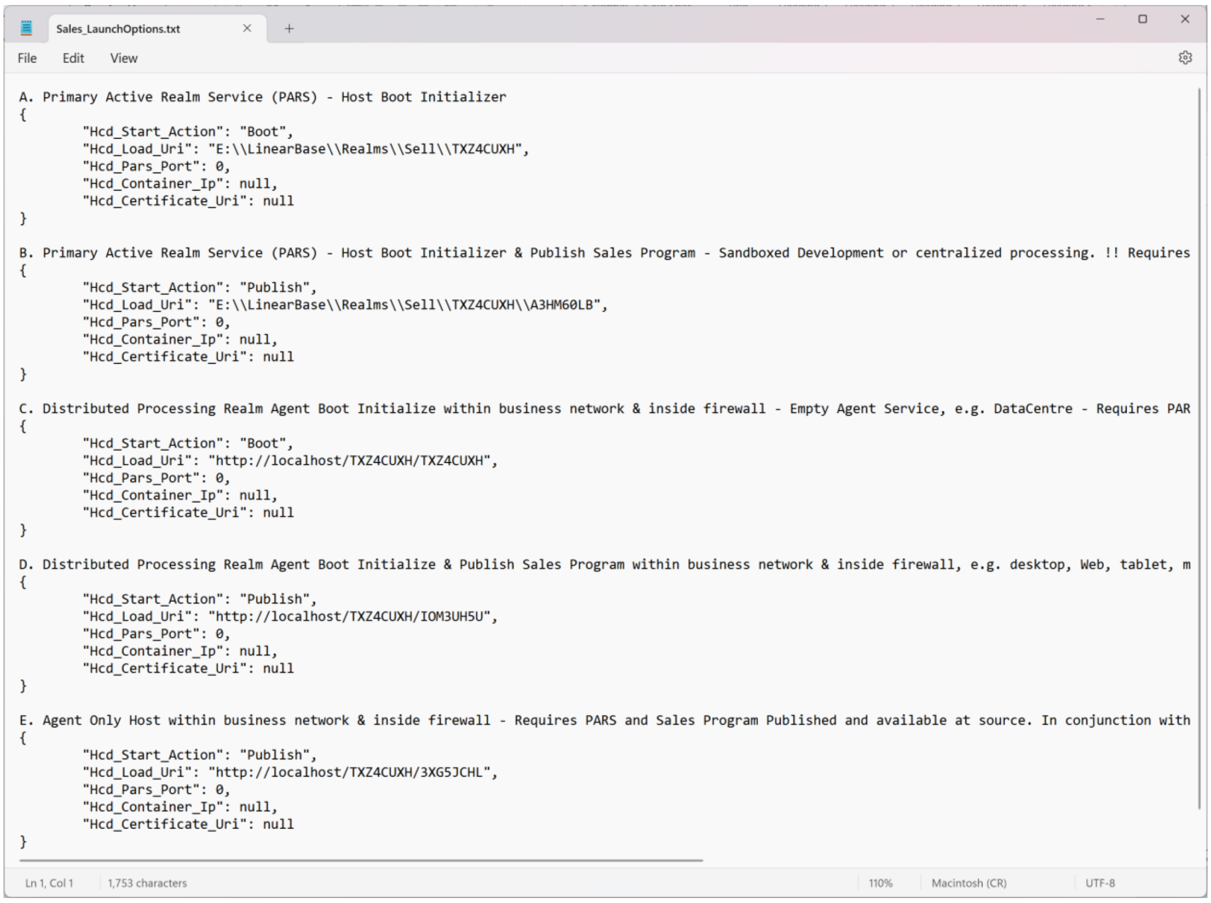


```
A. Primary Active Realm Service (PARS) - Host Boot Initializer
hcd.Boot(new Uri(@"E:\LinearBase\Realms\Sell\TXZ4CUXH"), 0, CancellationToken.None);

B. Primary Active Realm Service (PARS) - Host Boot Initializer & Publish Sales Program - Sandboxed Development or centralized processing. !! Requires P
hcd.Publish(new Uri(@"E:\LinearBase\Realms\Sell\TXZ4CUXH\A3HM60LB"), 0, CancellationToken.None);

C. Distributed Processing Realm Agent Boot Initialize within business network & inside firewall - Empty Agent Service, e.g. DataCentre - Requires PARS.
hcd.Boot(new Uri(@"http://localhost/TXZ4CUXH/TXZ4CUXH"), 0, CancellationToken.None);

D. Distributed Processing Realm Agent Boot Initialize & Publish Sales Program within business network & inside firewall, e.g. desktop, Web, tablet, mob
hcd.Publish(new Uri(@"http://localhost/TXZ4CUXH/IOM3UH5U"), 0, CancellationToken.None);

E. Agent Only Host within business network & inside firewall - Requires PARS and Sales Program Published and available at source. In conjunction with B
hcd.Publish(new Uri(@"http://localhost/TXZ4CUXH/3XG5JCHL"), 0, CancellationToken.None);
```

Figure 6 - Sales Initializers

Initializers are for use in AddService.cs extension file added to solution folder. It is possible to include more than one Profile, and therefore Program, at startup by adding multiple Publish instructions.

Launch Options



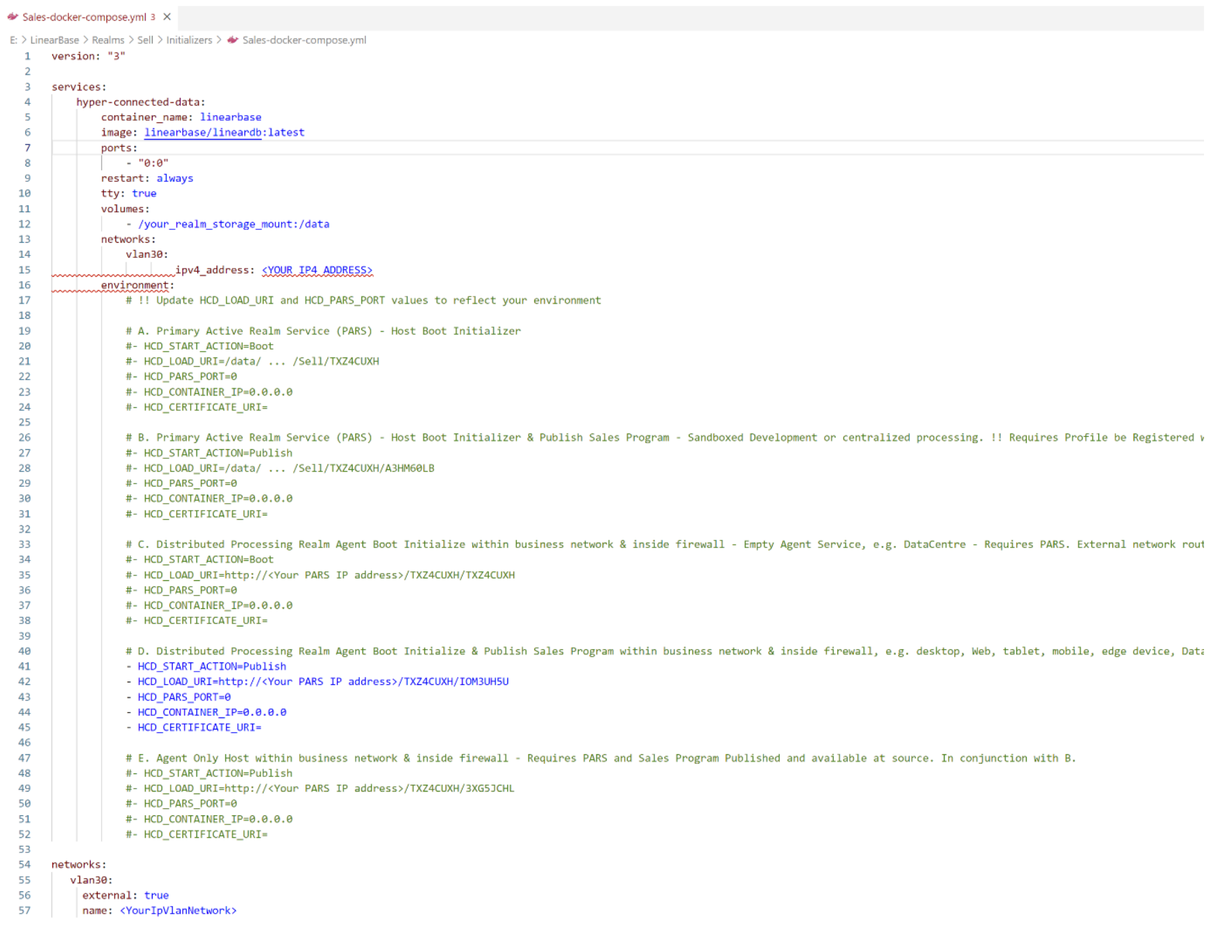
```
A. Primary Active Realm Service (PARS) - Host Boot Initializer
{
        "Hcd_Start_Action": "Boot",
        "Hcd_Load_Uri": "E:\\LinearBase\\Realms\\Sell\\TXZ4CUXH",
        "Hcd_Pars_Port": 0,
        "Hcd_Container_Ip": null,
        "Hcd_Certificate_Uri": null
}

B. Primary Active Realm Service (PARS) - Host Boot Initializer & Publish Sales Program - Sandboxed Development or centralized processing. !! Requires
{
        "Hcd_Start_Action": "Publish",
        "Hcd_Load_Uri": "E:\\LinearBase\\Realms\\Sell\\TXZ4CUXH\\A3HM60LB",
        "Hcd_Pars_Port": 0,
        "Hcd_Container_Ip": null,
        "Hcd_Certificate_Uri": null
}

C. Distributed Processing Realm Agent Boot Initialize within business network & inside firewall - Empty Agent Service, e.g. DataCentre - Requires PAR
{
        "Hcd_Start_Action": "Boot",
        "Hcd_Load_Uri": "http://localhost/TXZ4CUXH/TXZ4CUXH",
        "Hcd_Pars_Port": 0,
        "Hcd_Container_Ip": null,
        "Hcd_Certificate_Uri": null
}

D. Distributed Processing Realm Agent Boot Initialize & Publish Sales Program within business network & inside firewall, e.g. desktop, Web, tablet, m
{
        "Hcd_Start_Action": "Publish",
        "Hcd_Load_Uri": "http://localhost/TXZ4CUXH/IOM3UH5U",
        "Hcd_Pars_Port": 0,
        "Hcd_Container_Ip": null,
        "Hcd_Certificate_Uri": null
}

E. Agent Only Host within business network & inside firewall - Requires PARS and Sales Program Published and available at source. In conjunction with
{
        "Hcd_Start_Action": "Publish",
        "Hcd_Load_Uri": "http://localhost/TXZ4CUXH/3XG5JCHL",
        "Hcd_Pars_Port": 0,
        "Hcd_Container_Ip": null,
        "Hcd_Certificate_Uri": null
}
```

Figure 7 - Sales Launch Options

LinearBase.Host.exe initially checks for the existence of command lines before checking for the launch options configuration file launchOptions.json.

Navigate to the folder containing LinearBase.Host.exe and open file launchOptions.json. Change the port number of the relevant Profile instruction to between 2000 and 65535. Copy and paste the instruction into launchOptions.json overwriting the existing values. Save the changes. Select LinearBase.Host.exe and run the program by right clicking and selecting ‘Open’ from the context menu or double clicking or double tapping the program.

Docker Compose


Sales-docker-compose.yml 3
E: > LinearBase > Realms > Sell > Initializers > Sales-docker-compose.yml

```yaml
version: "3"

services:
    hyper-connected-data:
        container_name: linearbase
        image: linearbase/lineardb:latest
        ports:
            - "0:0"
        restart: always
        tty: true
        volumes:
            - /your_realm_storage_mount:/data
        networks:
            vlan30:
                    ipv4_address: <YOUR_IP4_ADDRESS>
        environment:
            # !! Update HCD_LOAD_URI and HCD_PARS_PORT values to reflect your environment

            # A. Primary Active Realm Service (PARS) - Host Boot Initializer
            #- HCD_START_ACTION=Boot
            #- HCD_LOAD_URI=/data/ ... /Sell/TXZ4CUXH
            #- HCD_PARS_PORT=0
            #- HCD_CONTAINER_IP=0.0.0.0
            #- HCD_CERTIFICATE_URI=

            # B. Primary Active Realm Service (PARS) - Host Boot Initializer & Publish Sales Program - Sandboxed Development or centralized processing. !! Requires Profile be Registered
            #- HCD_START_ACTION=Publish
            #- HCD_LOAD_URI=/data/ ... /Sell/TXZ4CUXH/A3HM60LB
            #- HCD_PARS_PORT=0
            #- HCD_CONTAINER_IP=0.0.0.0
            #- HCD_CERTIFICATE_URI=

            # C. Distributed Processing Realm Agent Boot Initialize within business network & inside firewall - Empty Agent Service, e.g. DataCentre - Requires PARS. External network rou
            #- HCD_START_ACTION=Boot
            #- HCD_LOAD_URI=http://<Your PARS IP address>/TXZ4CUXH/TXZ4CUXH
            #- HCD_PARS_PORT=0
            #- HCD_CONTAINER_IP=0.0.0.0
            #- HCD_CERTIFICATE_URI=

            # D. Distributed Processing Realm Agent Boot Initialize & Publish Sales Program within business network & inside firewall, e.g. desktop, Web, tablet, mobile, edge device, Dat
            - HCD_START_ACTION=Publish
            - HCD_LOAD_URI=http://<Your PARS IP address>/TXZ4CUXH/IOM3UH5U
            - HCD_PARS_PORT=0
            - HCD_CONTAINER_IP=0.0.0.0
            - HCD_CERTIFICATE_URI=

            # E. Agent Only Host within business network & inside firewall - Requires PARS and Sales Program Published and available at source. In conjunction with B.
            #- HCD_START_ACTION=Publish
            #- HCD_LOAD_URI=http://<Your PARS IP address>/TXZ4CUXH/3XG5JCHL
            #- HCD_PARS_PORT=0
            #- HCD_CONTAINER_IP=0.0.0.0
            #- HCD_CERTIFICATE_URI=

networks:
   vlan30:
      external: true
      name: <YourIpVlanNetwork>
```

Figure 8 - Sales Docker Compose

Docker is out of scope for this article however the file docker-compose.yml contains the relevant environment variables.

As this release reuses the specified PARS port across associated Realm initializations the Docker setup requires the use of a VLAN network to host images on individual IP addresses. A future release will include the Profile Manager Portal to allow for custom port assignment.

Model

To create a new Databank with associated code without using an existing database read the article the How-To Digitally Model a New Databank Program and Code.

Next Steps

Develop enterprise scale data driven software solutions without the need for traditional databases. LinearBase support simultaneous deployments across Windows, Android, iOS, Mac and Linux, with Docker-on-Linux for servers.

Add inline distributed relational data processing client functionality and server capability to any .NET application by:

1. add the AddService.cs file, contained in the Extension.zip folder, to the project solution folder
2. initiate the service by adding Add???() to the program.cs ConfigureServices section, in the example this is AddSales()
3. include a reference in the .csproj file to one of the two LinearBase NuGet packages
    a. LinearBase.Server, does not support Android in this release
    b. LinearBase.Client, does not listen for requests from other Clients

Finally, help us improve the software by giving feedback to feedback@linearbase.com