

Embedded Database Structures with Built Remote Clients

By Tai Bui, Technical Services Engineer, 4D Inc.

Technical Note 25-03

Table of Contents

Table of Contents	2
Abstract	4
Introduction	4
4D Deployment Options	4
4D Application Deployments.....	4
Database Structure and Execution Modes	4
Merged (Built) Applications	5
Limitations of Standard 4D Remote Client Execution.....	5
Embedded Database Feature	5
Process Overview	6
Connecting to a 4D Server	6
Building the 4D Remote Client with the Embedded Database	6
Post-Build Process.....	6
Example Implementation.....	7
Overview of Implementation	7
Embedded Standalone Database	7
Client Server Database	8
Demo Steps	8
Demo Structure for Embedded Database	8
Step 1: Add 4D Internet Commands to the Database	8
Step 2: Open the Standalone Database.....	9
Optional Step: Exploring the Source Code	9
Step 3: Open Build Window.....	9
Step 4: Configure the Build Settings to Build Compiled Structure.....	10
Step 5: Generate the Compiled Structure.....	10
Demo Structure for Client Server Database	11
Step 6: Open Client Server Database	11
Optional Step: Exploring the Source Code	11
Step 7: Open Build Window.....	11
Step 8: Built server application settings.....	11
Step 9: Built client application settings.....	11
Step 10: Generate the "buildApp.4DSettings" file	12
Step 11: Update Settings to Embedding the Compiled Standalone Database	12
Step 12: Generate the Built Client and Server Applications.....	13
Step 13: Add 4D Internet Commands to built remote Client Application	13
Running the Demo.....	13
Step 15: Create a Copy of the Built Server Package and Run one of them	13
Step 16: Open the User Settings Window	13
Step 17: Change the Port Number	14
Step 18: Start up the Second Server	14
Step 19: Navigate through the Built Client Applications Directory	14
Step 20: Create .4dlink files to Specify Servers and Priority	15
Step 21: Run Built Remote Client Application.....	15
Step 22: Close the Client and "Primary" Server.....	15
Step 22: Run Built Remote Client Application Again	15
Conclusion	16

Abstract

A built 4D Client-Server application provides an efficient way to deploy a 4D database application to multiple users. The compiled application ensures source code security and gives developers control over the user experience. However, a common limitation of 4D Client-Server deployment is that a 4D Remote Client cannot execute developer-written code until it successfully connects to a 4D Server.

To address this, 4D allows a built 4D Remote Client to start with a standalone database, enabling it to execute code without an initial server connection. This Technical Note explores the process of embedding a standalone 4D database within a 4D Remote Client. Additionally, it presents a sample implementation demonstrating how a 4D Remote Client can automatically connect to a primary server and failover servers.

Introduction

This Technical Note explores 4D's built client-server application feature, which allows a standalone database to be embedded within the built client. This capability enables the client to execute code independently before connecting to a 4D server.

The document will first explain the key concepts and details of this feature. It will then provide a step-by-step implementation example demonstrating an automatic failover process, where clients automatically connect to a secondary "failover" server if the primary server is unavailable. Sample databases for these examples are included with this Technical Note.

4D Deployment Options

4D provides multiple ways to run a database. This section offers a brief overview of these deployment methods to provide context for the feature discussed in this Technical Note. A running 4D database application consists of two main components:

The 4D Application – The runtime environment.

The 4D Database Structure – The code and logic of the application.

Depending on the deployment strategy, these components can be configured in different ways.

4D Application Deployments

4D Standalone - A 4D application runs as an independent instance on a single machine, containing the full database application within a self-contained environment. This is performed by using the 4D application and opening the database structure directly. This method is ideal for individual use or non-networked scenarios where collaboration is unnecessary.

4D Client-Server – This deployment supports multi-user collaboration by hosting the database on a 4D Server application. Multiple 4D Remote Clients can then connect to this centralized server, enabling shared access to the database.

Database Structure and Execution Modes

The database structure defines the logic and functionality of a 4D application and can be deployed in different modes and forms:

Interpreted Mode – Runs directly from the source code, allowing real-time modifications and debugging. This mode can be slower as the machine must “interpret” the human written code. This is available from a binary .4DB structure file and .4DProject structure format.

Compiled Mode – Precompiles the code into machine language, improving execution speed and security. This can be available in .4DB and .4DProject structure formats if the code has been compiled and included. This mode is also the only option for a compiled structure package which only provides the compiled code and removes the human written code. This also enhances security of the code.

Merged (Built) Applications

A merged or built application integrates the 4D application with the compiled database structure, creating a self-contained executable. This eliminates the need for a manual startup process, providing a seamless user experience and greater control over deployment.

Limitations of Standard 4D Remote Client Execution

When a 4D application starts, it does not initially execute any custom code. Instead, it requires one of the following actions:

- Opening a database structure, or
- Connecting to a 4D Server.

In a built/merged 4D Remote Client, the application automatically attempts to connect to its designated 4D Server upon launch. However, until a successful connection is established, no custom-written code can be executed.

If the client fails to connect, it typically reports the issue and quits by default. Alternatively, it can be configured to prompt the user to select a server through the remote connection dialog. However, even in this scenario, custom code cannot run until a connection is successfully established.

Choosing the right deployment method for a 4D database depends on factors such as collaboration needs, performance considerations, and security requirements. While a standalone application provides a self-contained solution, a client-server deployment enables centralized data access for multiple users. Additionally, utilizing compiled and merged applications enhances performance and protects source code. Understanding these deployment options sets the foundation for leveraging advanced 4D features, such as embedding a standalone database in a remote client, which is explored further in this Technical Note.

Embedded Database Feature

A built 4D client-server application is an excellent choice for deploying a 4D database application. However, as discussed in the previous section, there is a limitation where custom-

written code cannot be executed in a built 4D client until it successfully connects to a server. To address this limitation, 4D provides a feature that allows a standalone database structure to be embedded directly into the built remote client.

When a standalone database structure is embedded into the client, the client will first open the standalone database and run it in standalone mode. This allows custom code from the embedded database to be executed without requiring the client to connect to a 4D server. This feature offers greater flexibility and control over the user experience for deployed 4D remote clients.

Process Overview

To implement this feature, there are a few key steps to follow. First, you need two 4D structures: one for the client-server database and another for the standalone database that the 4D remote client will open initially. The standalone database must be a compiled structure (.4DC or .4DZ) and should reside on the same machine as the client-server structure when the build is performed.

Connecting to a 4D Server

The embedded standalone database does not require a data file, so there's no need for the user to define or create one. To connect to the 4D Server, the standalone database must call the **OPEN DATABASE** command, passing a 4D access file (.4dlink) that contains the 4D Server's address. Upon executing this command, the transition from the standalone structure to the 4D server should happen seamlessly, with no unexpected behavior.

Building the 4D Remote Client with the Embedded Database

To merge the standalone database with the 4D remote client, you must configure the 4D Build XML file. It is recommended to start with a default build XML file, which can be generated using the Build Wizard in developer mode. Once the desired build options are set, you can save the configuration, which will generate a baseline Build XML file.

While the option to embed a standalone database is not available directly through the Build Wizard, it can be manually added to the Build XML file. Open the file and add the following XML keys based on the target deployment OS:

- **Mac:**
/Preferences4D /BuildApp / SourcesFiles / CS
/DatabaseToEmbedInClient**Mac**Folder
- **Windows:**
/ Preferences4D / BuildApp / SourcesFiles / CS /
DatabaseToEmbedInClient**Win**Folder

For the value of the key, specify the full path to the folder containing the compiled standalone structure. After adding the necessary key, the final step is to perform the build process.

Post-Build Process

Once the build succeeds, the resulting merged 4D Server database application will function as the typical server application. The 4D Remote Client, on the other hand, will be packaged with the embedded compiled standalone structure. It's important to note that while it's possible to develop a single structure that can act as both the standalone and client-server database, this approach is not recommended. The size of the structure would increase the client machine's footprint, as it would include both the embedded standalone database and cached files from the server database.

When the 4D Remote Client is launched, the compiled standalone structure will open first instead of attempting to connect to the 4D Server. This eliminates the risk of failure due to network or connection issues and allows custom code to execute once the standalone structure opens.

This process allows a merged 4D Remote client to execute written code without the need to successfully connect to the 4D Server first by embedding a compiled 4D structure. The application can run the structure locally and execute code. When the application is ready to connect to a 4D server, a call to **OPEN DATABASE** can be made, allowing the database to transition to a remote client and connect to the 4D Server.

Example Implementation

The embedded database feature can be used in many ways. Some possible implementations are to provide a custom connection portal for users to select a specific server to connect to, to run some checks on the system, or, in the case of this Technical Note, to provide an automatic failover feature. The code used for the implementation is simple and is not very strict. It is intended to get the point across and provide an understanding of the embedded database feature and a proof of concept for an automatic failover feature.

Overview of Implementation

The automatic failover feature will be implemented as such:

- 1) The standalone database will start up and search a directory for a list of numbered .4Dlink files in order of priority.
- 2) If the files are found, the server address in the file will be checked to see if it is running.
- 3) If the server is not available, it will check the next .4dlink file found.
- 4) When it finds one that is available it will connect to it using **OPEN DATABASE** with the file.
- 5) If no servers are found, the remote client will attempt a default attempt on any available database found on the default port 19813.
- 6) If all fails, an error message will be displayed.

Embedded Standalone Database

The standalone database, named "Demo_EmbeddedDB", contains the main procedures of this example. The standalone database's purpose is to allow the 4D remote client to automatically connect to failover servers if a primary server is not available for some

reason. This can allow users a more elegant experience by allowing them to continue their work without disruption nor having an error message displayed. The goal of an embedded database is to connect to a server successfully using **OPEN DATABASE** allowing the 4D Remote client running a standalone database to transition to a 4D remote client connected to a 4D server database.

The checking of the server is performed by using a UDP check with the 4D Internet Commands plugin. A nuance of the use of plugins in an embedded database structure to a built remote client is that the client-server database needs to host the plugin, and the built remote client must have the plugin.

Client Server Database

The 4D server database will be a simple database that simply displays an alert confirming the connection on the 4D remote client. The client-server database enables the user settings. This allows the settings to be modified in a built database. Multiple copies of the built 4D server can be run on different ports. This will be used as a way to replicate multiple servers running as a primary and failover for the demo.

The implementation described has been created with the associated demo database. The source is available to explore and follows the overview described. The demo does not provide the 4D Internet Commands and will need to be added using the one appropriate for the version and build of 4D used.

Demo Steps

This section describes the suggested steps to perform to explore the feature of embedding a standalone database into a built remote 4D Client using the associated demo database. Due to the nature of how to implement the feature, through performing a build in a specific way, the demo and steps cannot be easily self-contained in a single database. The process of the demo relies on the default layout of the packaged demo with the Technical Note. There should be a demo directory as part of the Technical Note package.

Demo Structure for Embedded Database

In the demo directory, the standalone database is in the directory named "Demo_EmbeddedDB".

Step 1: Add 4D Internet Commands to the Database

First, the 4D Internet Commands plugin needs to be added to the database. Go to the installed directory for the 4D applications. There should be a "Plugins" directory containing the "4D InternetCommands.bundle" plugin. Copy the plugin to the "Plugins" directory located in the same level as the database's "Project" directory.

If the 4D Internet Commands plugin is not found with the installed 4D package (the case for more recent R releases), it can be found on the download site at:

product-download.4d.com

Step 2: Open the Standalone Database

First, open the standalone database with the standard 4D application for development. When it opens, the window should display, providing a summarized description of the database structure and the steps to perform. The code is provided in the database structure and can be viewed.

Optional Step: Exploring the Source Code

The process starts with the On Startup Database method, which calls the “**Startup**” project method. The “Startup” project method checks the environment of how the database is running. The database uses a check for if the application type running the database is 4D is [4D Volume desktop](#), which is the type used when running a built remote client. If it is not, it will display the demo description window. If it is then it will run the “**findServer**” method.

The “**findServer**” method is used to find, check, and connect to the highest priority Server running. First, the method checks if the 4D Internet Commands plugin is available. If not, it will default to attempting to connect to the primary server without checking if it is available which can cause an error.

If the 4D Internet Commands plugin is available, it will parse through the .4dlink files found in the database’s /Resources/Servers/ directory in sorted file name order checking if the server is available. The first server found to be available will be connected to using **OPEN DATABASE** on the .4dlink file.

As mentioned, the code is simple, and the priority is based on sorted file names. For the sake of keeping the demo simple, it is recommended to name the .4dlink files with 1-9, such as “1.4dlink”, “2.4dlink”, or “6.4dlink”.

If all .4dlink files are parsed and no available server is found, a .4dlink file will be generated with a default attempt on the first server it can find under the default port of 19813. This can cause an error if no server is found, or an incorrect server is connected to.

The “**pingServer**” method is used to check if the server is available through UDP using the 4D Internet commands. This method is based on the code from the Documentation.

Step 3: Open Build Window

When ready, build the compiled structure by going to the menu bars “Design” menu and selecting “Build Application...”

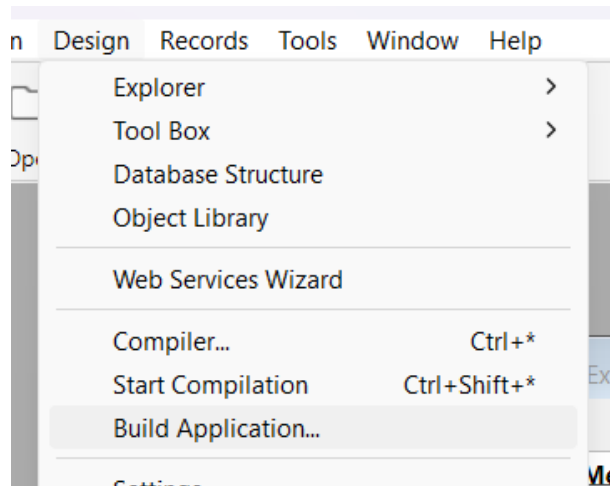


Image 1: Design Menu with Build Application item towards bottom.

A message might display requiring the code to be compiled. Accept, if it displays, to compile the code.

Step 4: Configure the Build Settings to Build Compiled Structure

Make sure that the Compiled structure is generated by going to the “Compiled structure” tab and enabling the “Build compiled structure” option. Also, make sure to note or update the “Destination Folder” at the top of the window as the path is needed for the embedding process.

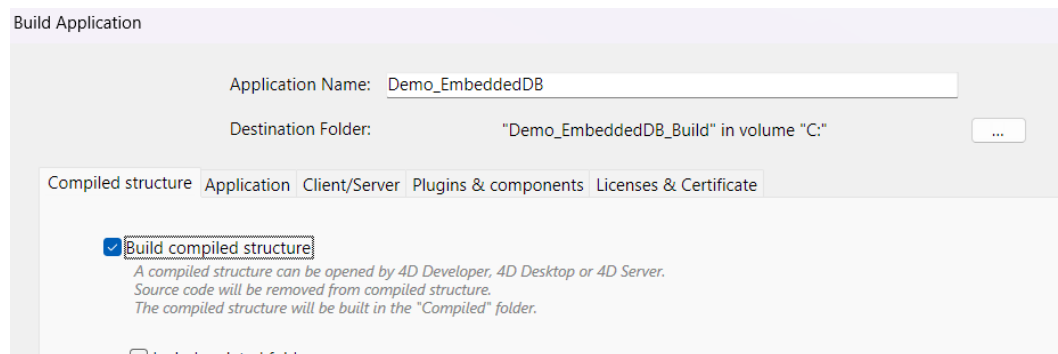


Image 2: Build compiled structure option enabled.

Step 5: Generate the Compiled Structure

After the options are specified, the “Build” button on the lower left can be clicked to perform the build. After the build process succeeds, the database can be closed. By default, the build is generated next to the database’s directory appending an “_Build” to the new directory. Inside the build directory the compiled database structure should be located inside a “Compiled Database” structure with the name of the database. The default path based on the “Demo” directory should be:

...\Demo\Demo_EmbeddedDB_Build\Compiled Database\Demo_EmbeddedDB\

Demo Structure for Client Server Database

Next, the client-server database is in the “Demo_CS” directory.

Step 6: Open Client Server Database

Afterwards the database can be opened in the standard 4D application for development. Similar to the “Demo_EmbeddedDB” structure, the structure should display a window summarizing the structure and steps to perform.

Optional Step: Exploring the Source Code

The database’s On Startup Database method calls the “**Startup**” method. This method checks the application type to assume and perform the following. If the type is [4D Volume desktop](#), it is assumed that the database is running as a 4D Remote client. In this case, it will report the Port number that the Server using. If the application type is 4D Local mode, it is assumed that the database is running in single-user interpreted mode. In all other cases, it is assumed that it is possibly running as a server so there is nothing to perform.

Step 7: Open Build Window

When ready, go to the build application window to configure the Client/Server build settings. Like previously, if a compilation is needed perform it.

Step 8: Built server application settings

This time, go to the “Client/Server” tab. Enable the “Build server application” option at the top and specify the 4D Server location with the location of the installed 4D Server directory. Setting the “Data linking mode based upon the” “Application path” will help the demo as it will allow two instances of the server to run easier.

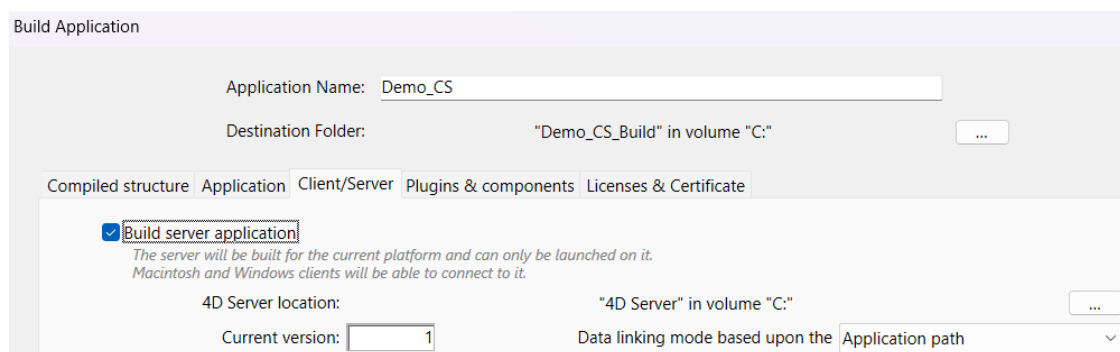


Image 3: Build server application setting enabled with 4D Server location specified

Step 9: Built client application settings

Then enable the “Build client application” option below the server settings and specify the location of the installed 4D Volume desktop location.

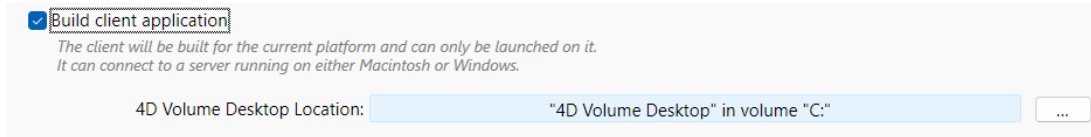


Image 4: Build client application setting enabled with 4D Volume Desktop location specified

Step 10: Generate the "buildApp.4DSettings" file

Afterward, go to the bottom left and click the "Save settings" button. This will generate a "buildApp.4DSettings" file with the applied settings in the databases "Settings" directory located next to the "Project" and "Plugins" directories. The Build Wizard can be closed now.

Step 11: Update Settings to Embedding the Compiled Standalone Database

Now the previously compiled structure will be set to be embedded in the built remote client application. This must be performed manually as this option is not listed in the Build window. Open the "buildApp.4DSettings" file with a text editor, preferably one that formats xml.

Start from <Preferences4D>, then <BuildApp>, <SourcesFiles>, and then <CS>. The order of the keys in the <CS> section does not matter so as long as it is not added within another key. The pattern of keys can be added by entering the pattern:

```
< {key} > {value} </ {key}>
```

Make sure that the key is closed with </ {key} >. As a reminder, the attribute for Mac is **DatabaseToEmbedInClientMacFolder** and Windows is **DatabaseToEmbedInClientWinFolder**. The value should be the full path to the compiled structure package.

For example, on Windows it could look something like:

```
...
<Preferences4D>
...
<BuildApp>
...
<SourcesFiles>
...
<CS>
...
  <DatabaseToEmbedInClientWinFolder> C:\Demo\Demo_EmbeddedDB_Build\Compiled
  Database\Demo_EmbeddedDB </DatabaseToEmbedInClientWinFolder>
...
</CS>
...
</SourcesFiles>
...
</BuildApp>
...
```

```
</Preferences4D>
```

```
...
```

Save the changes and then go back to the Build Wizard and perform the build without making any changes to the file from the wizard that might remove the added key.

Step 12: Generate the Built Client and Server Applications

Open the build wizard again and click on the “Build” button. This will generate the built application with a Built Server and a Built Remote Client application. Similarly to the compiled structure, the default location of the generated files should be next to the database directory with a “_Build” added to the new directory’s name. In the directory, there should be a “Client Server executable” directory containing two more directories for the “Client” and “Server”.

Step 13: Add 4D Internet Commands to the Built Remote Client Application

The 4D Internet Commands also needs to be added to the built remote Client Application. As mentioned, a nuance of an embedded database is that it loads the plugins from the host-built database. The plugins were added to the embedded database structure earlier to allow compilation to succeed. For a client-server application, the plugins are typically added to the server and then cached and loaded on the client side when it connects to the server. Add the plugin in a “Plugins” directory in the client directory.

On Windows, this will be next to the executable of the built client application. On Mac, this will be in the contents of the application, which can be viewed by performing a contextual click on the application and selecting view contents.

Running the Demo

Now there should be a built server and a built client with an embedded standalone database. The following steps can be performed to simulate running a primary and failover server to see how the standalone database’s code executes prior to the client connecting to anything allowing it to automatically find an available server and connect to it.

Step 15: Create a Copy of the Built Server Package and Run one of them

First, create a second copy of the “Server” directory. Then run one of the built server applications.

Step 16: Open the User Settings Window

Go to the database’s User settings under the menu bar’s “Edit” menu > “Settings” > “User Settings...” or “User settings for Data file...” item.

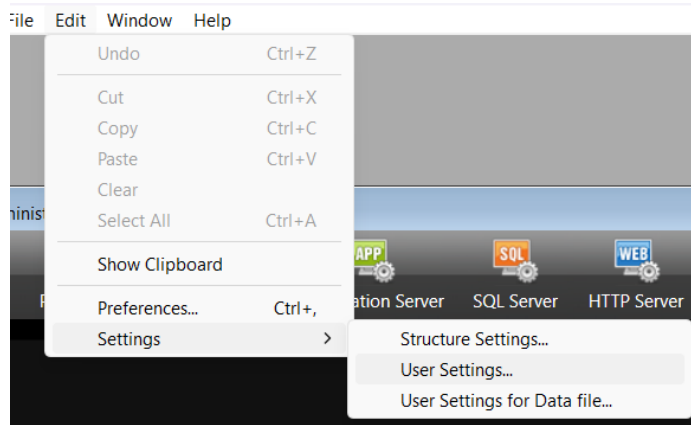


Image 6: Settings menu items

Step 17: Change the Port Number

In the settings window, go to the “Client-server” section > “Network options” tab. The Port Number is defined here to be using the default 19813. Change this to another available port such as something like 19823. To prevent an error message, it is also suggested to change the SQL port to something like 19822. Restart the Server application to apply the changes.

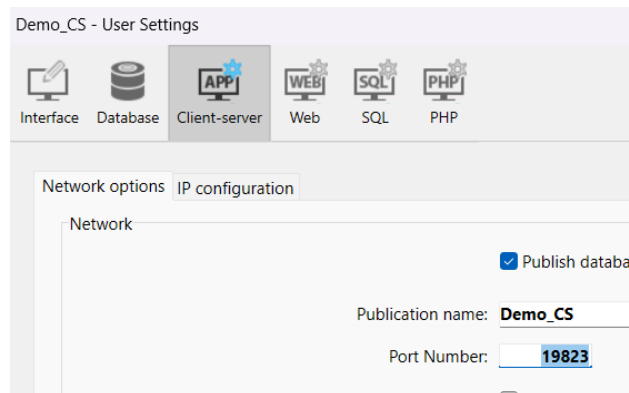


Image 7: Port Number Setting

Step 18: Start up the Second Server

Start up the second server. This should be running on the default port number. If there are any conflicts with port numbers, feel free to use a different number.

Step 19: Navigate through the Built Client Applications Directory

Now go to the built client directory containing the executable and enter the “Database” directory. Typically, this directory contains a .4dlink file associated with the built 4D Server, however, because a database was embedded, this directory contains a compiled structure (.4DZ) which is essentially a renamed copy of the compiled embedded database generated at the start.

The directory also contains a “Resources” directory. Enter this directory and create a “Server” directory.

Step 20: Create .4dlink files to Specify Servers and Priority

A template for .4dlink files can be found in the Technical Note’s \Demo\templates\ directory. The template should only need the address of the server and the port numbers updated. The template applies the localhost IP and can be used as is.

Create a “1.4dlink” file and a “2.4dlink” file with the information for each of the two servers running. The server associated with “1.4dlink” will act as the “Primary” server while the second one will act as the “Failover” server. For example, the “Primary” server can be the first 19823 server while the “Failover” server can be the 19813 server.

Step 21: Run Built Remote Client Application

Now the built remote client can be run. The client should launch in standalone mode with the embedded database structure (Demo_EmbeddedDB) and run its **“Startup”** method. The process should proceed with the application attempt to connect to the server defined in “1.4dlink”. The client should successfully connect to it and the **“Startup”** method for the Demo_CS structure should run displaying a dialog with the Port number of the server.

Step 22: Close the Client and "Primary" Server

Now close the client. Then close the Server that was acting as the “Primary” server referenced in “1.4dlink” but leave the second server running.

Step 22: Run Built Remote Client Application Again

Run the built remote client again and this time it should parse “1.4dlink” and not find the server. It should then iterate to the next .4dlink file to find and connect to the “2.4dlink” server. The dialog should then display the port number of the second server.

The sample database provides an example implementation that utilizes the feature that allows a standalone structure to be embedded to a built 4D remote client. While it might have seemed like there were a lot of steps, the process is actually not too complicated.

The embedded structure is a standalone database that can run code prior to a server connection and should end up with a call to **OPEN DATABASE** on a .4dlink file. A compiled structure for the embedded database is generated and then embedded to the client/server database application through the build settings.

The demo expands on the feature and provides a proof of concept of a use case for this feature that allows a built 4D remote client to automatically search and connect to a server based on levels of priority.

Conclusion

This Technical Note has explored the powerful feature of embedding a standalone database into a built 4D remote client, offering a robust solution for deploying 4D database applications. By embedding a compiled standalone structure directly within the client, developers can execute code without requiring an initial connection to a 4D server, thereby improving user experience and flexibility. Through the implementation of an automatic failover system, it was demonstrated how this feature can ensure uninterrupted access to a 4D server, even if the primary server becomes unavailable. This failover mechanism provides a seamless transition between servers, enhancing the reliability of remote client connections. This feature enhances the adaptability and resilience of 4D client-server applications and provides new possibilities for application design, control, and user experience.