



# Deploying the openPDC on POSIX Platforms

## [Contents](#)

Introduction .....	3
Installation Prerequisite: Mono .....	4
Installing the openPDC .....	5
Troubleshooting .....	5
Running the openPDC .....	6
Execution as Terminal Application .....	6
Execution as Daemon .....	6
Controlling the Daemon .....	7
Running the Remote Console .....	7
Testing the Default Configuration .....	9
IEEE C37.118 Output Stream .....	9
Gateway Exchange Protocol .....	9
Configuring the openPDC .....	10
Manually Updating Configuration using SQL .....	10
Updating Configuration Using the openPDC Manager .....	10
Installing the openPDC Manager .....	11
Using the Default SQLite Configuration .....	12
Copy SQLite Configuration for Local Update .....	12
Use SQLite Configuration via File Share .....	13
Using Other Databases .....	14
Enabling the openHistorian .....	15
Completing Security Configuration .....	16
Managing User Accounts .....	17
Adding New Users .....	17
Authenticating Cross Platform Users .....	17
Disabling Pass-Through Authentication .....	18
Updating Transport Layer Security .....	19
Manually Creating a Certificate .....	19
Using an Existing Certificate .....	20
Changing the TLS Version .....	20
Running the openPDC with Elevated Rights .....	21
Securing the Configuration Database .....	21
Running on a Raspberry Pi .....	23



## Introduction

With the release of version 2.1 of the [openPDC](#), GPA has implemented features that allow the openPDC to easily be used in deployments running a POSIX<sup>1</sup> compatible operating system such as Linux and Apple's Mac OS X. GPA has tested several common distributions and hardware combinations that include:

- Debian on x86 style hardware (64-bit)
- Ubuntu on HyperV (64-bit)
- Ubuntu on SEL-3354 hardware (32-bit)
- Raspbian on Raspberry Pi and Pi 2 hardware (32-bit)
- Mac OS X on MacBook Air hardware (32-bit)

This document describes the process to install, configure and operate the openPDC on a POSIX compatible operating system. The directions that follow focus on Linux deployments of the openPDC; however, where necessary, there are exceptions noted that may be needed for Mac OS X deployments.

---

<sup>1</sup> POSIX, Portable Operating System Interface, is a family of IEEE standards to assure compatibility among operating systems. Most distributions of Linux, Mac OS X and iOS as well as Android are POSIX compatible.



## Installation Prerequisite: Mono

The openPDC is a .NET application and as such it requires a Common Language Runtime (CLR) engine to execute. For use in POSIX compatible environments, the openPDC uses [Mono](#) as its runtime engine. Mono is a cross platform implementation of the .NET framework that is widely available on various hardware and software architectures. For the openPDC version 2.1, Mono version 3.12 is required<sup>2</sup>.

To install Mono, follow the installation guide using Xamarin packages for your distribution:

<http://www.mono-project.com/download/>

Once Mono is installed, validate that the system is running the proper version of Mono using the **mono** application with the version parameter, **-V**, from a terminal session. The reported version should be “version 3.12” (or later) as shown below.

```
$ mono -V

Mono JIT compiler version 3.12.1 (tarball Tue Mar 17 20:43:02 UTC 2015)
Copyright (C) 2002-2014 Novell, Inc, Xamarin Inc and Contributors. www.mono-project.com
  TLS:             __thread
  SIGSEGV:         altstack
  Notifications:   epoll
  Architecture:    x86
  Disabled:        none
  Misc:            softdebug
  LLVM:            supported, not enabled.
  GC:              sgen
```

If after installation of the latest version of Mono the **mono** application reports an older version<sup>3</sup>, modify the system path so that version 3.12 of Mono is found first.

If there are no Mono version 3.12 packages available for your POSIX distribution, then Mono must be compiled manually. In this case, follow these instructions:

<http://www.mono-project.com/docs/compiling-mono/>

---

<sup>2</sup> It is possible that an older version of Mono may work, but the openPDC version 2.1 incorporates many features of .NET 4.5 that are only available in later versions of Mono. The openPDC has not been fully tested with earlier versions of Mono; therefore, use of Mono version 3.12 or later, e.g., version 3.12.1 that includes the [FREAK security fix](#), is strongly recommended.

<sup>3</sup> It is possible to have multiple versions of Mono simultaneously installed on a system.



## Installing the openPDC

Due to the vast number of POSIX operating systems and hardware combinations, there are currently no precompiled deployment packages for the openPDC<sup>4</sup>. To begin the process of installing the openPDC, download the following script file<sup>5</sup>:

<http://www.gridprotectionalliance.org/Products/openPDC/Scripts/install-openPDC.sh>

### Mac OSX Note:

The **wget** application must be available before running this script. To install wget, install Brew from <http://brew.sh> and then run:

```
brew install wget
```

Note that X11/XQuartz is not required for openPDC.

For a default installation and configuration, the script can be executed right away to get things started. However, to customize the installation or complete the security configuration, review the following installation options.

Below are the available script parameters to allow customization of the installation:

- n: Use nightly builds
- p: Preserve source code
- d: Install destination (defaults to /opt/openPDC/)
- u: Uninstall the openPDC

To fully enable security for this installation, as detailed later, preserve the local source code by using the **-p** parameter. To install the openPDC using the latest source code instead of the published stable release, use the **-n** parameter to install the latest nightly build.

The script should be run using the **bash**<sup>6</sup> shell with root access, for example, to install the openPDC and preserve the source code after installation run the following:

```
sudo bash install-openPDC.sh -p
```

## Troubleshooting Installation

If there are issues with the script during the installation process, validate that the Mono build system, XBuild, is working properly. To do this, execute the **xbuild** application with a version parameter, **/version**, from a terminal session and verify that the application reports version 12.0 of the XBuild Engine, similar to the following:

---

<sup>4</sup> As time allows, GPA will add deployment packages for common POSIX distributions, e.g., Debian and Ubuntu on Linux and Mac OS X. Third-parties are welcomed to assist with maintaining up-to-date deployment packages for the openPDC on various distributions.

<sup>5</sup> During execution the script will require internet access to download files to be installed. Since other scripts and source code files will be downloaded as part of the installation, it is best to run the script from its own folder.

<sup>6</sup> The **bash** shell command is required over the normal **sh** command to be able to properly test for Mac OS X platforms. Most all scripts to be executed, as referenced in this document, will require bash for this reason.



```
$ xbuild /version
XBuild Engine Version 12.0
Mono, Version 3.4.1.0
Copyright (C) 2005-2013 Various Mono authors
```

If the version of XBuild is not the same as reported above or the application fails to run, verify the path to the **xbuild** command is correct.

If the installation script needs to be run again, clear out the temporary files in the installation folder (i.e., the folder where script was run) and delete the destination folder (e.g., /opt/openPDC) before the next attempt.

## Running the openPDC

The installation script will automatically deploy the openPDC with a default configuration, as such the application can be run immediately. The openPDC can be started as a terminal application or registered as a daemon that will run in the background and startup automatically.

### Execution as Terminal Application

To test the openPDC or validate its current configuration, it can simply be run as a terminal application. To run the openPDC as a terminal application, execute the **openPDC.exe** application using the Mono runtime and the **RunAsConsole** parameter:

```
sudo mono /opt/openPDC/openPDC.exe -RunAsConsole
```

This will start the openPDC as a console style application with feedback reported directly to the current session. When running in this mode, input can be accepted just by typing. Note that keyboard input is being logged even if any user input is visually interrupted by a new status update from the openPDC. As an example to start issuing commands, just type **Help** and press enter for a list of possible commands.

To exit the application and stop the openPDC, just type the command **Exit** and press enter.

### Execution as Daemon

To run the openPDC automatically at startup, the application must be deployed to run as daemon in the background. To configure the daemon to run automatically startup, execute the registration script<sup>7</sup>:

```
sudo bash register-openPDC.sh
```

If the openPDC needs to be unregistered later, use the **-u** parameter with this script to unregister the service. Note that uninstalling the openPDC will also automatically unregister the service.

When running the openPDC as a background service, input and output from the application must be handled by another application, such as the openPDC Remote Console. See the *Running the Remote Console* section for more details.

---

<sup>7</sup> Note that needed scripts are downloaded as part of the install process. In order to run properly both the **openPDC.sh** and **register-openPDC.sh** scripts are required to be in the same folder during execution; for Mac OSX, this also includes the **openPDC.plist** file.



### Controlling the Daemon

To manually control the openPDC service running as a daemon, use the following commands:

#### *Start Service*

```
sudo /opt/openPDC/openPDC start
```

#### *Restart Service*

```
sudo /opt/openPDC/openPDC restart
```

#### *Pause Service*

```
sudo /opt/openPDC/openPDC pause
```

#### *Resume Service*

```
sudo /opt/openPDC/openPDC resume
```

#### *Stop Service*

```
sudo /opt/openPDC/openPDC stop
```

### Running the Remote Console

To view the current activity and issue commands to the openPDC while it is running as a daemon, run the openPDC Remote Console.

#### *Run Locally*

The remote console is already preconfigured to properly connect to the locally running openPDC when executed from the same machine. To start the remote console, execute the following command from a terminal session<sup>8</sup>:

```
mono /opt/openPDC/openPDCConsole.exe
```

#### *Run Remotely*

The openPDC can be accessed from a remote machine by running the remote console application on another machine, including from a Windows installation<sup>9</sup>. In order for the remote console application to properly connect to the machine running the openPDC, the machine's IP address or DNS name will be required. There are two options for specifying the openPDC server to connect to: one way is by using a command line parameter for the openPDCConsole application, the other is setting the configuration so that the openPDCConsole always connects to the desired remote machine.

---

<sup>8</sup> The remote console application does not require being run as the root user, i.e., no **sudo** required.

<sup>9</sup> Run the Windows installer for the openPDC to get the openPDC Remote Console running in a Windows environment; download from here: <https://openpdc.codeplex.com/releases>. The installed components can be reduced to only the manager and tools during the installation process by deselecting the openPDC Service option, see *Installing the openPDC Manager* section for more details.



### Specifying the openPDC Server on the Command Line

The openPDCConsole application will accept the openPDC server name using the **-server** command line parameter, for example:

```
mono /opt/openPDC/openPDCConsole.exe -server="openPDC:8500; interface=0.0.0.0"
```

### Specifying the openPDC Server in the Configuration

To configure the console application running on a remote machine to always connect to a specific openPDC service instance, make sure the remote console application is not running then edit<sup>10</sup> the **openPDCConsole.exe.config** file and find the following settings that need to be updated:

```
<?xml version="1.0"?>
<configuration>
  <categorizedSettings>
    <remotingClient>
      <add name="ConnectionString" value="server=openPDC:8500; interface=0.0.0.0" />
      <add name="IntegratedSecurity" value="False" />
    </remotingClient>
  </configuration>
```

The **openPDC** text, as shown in the **ConnectionString** or in the **-server** command line parameter example above, must be replaced with the DNS name or IP that is running the openPDC service. Do not forget to include “; interface=0.0.0.0” in the changes as this forces the connection to be IPv4<sup>11</sup>. If the openPDC Remote Console application is being used on a Windows machine to connect to the openPDC service running in a POSIX environment, also set **IntegratedSecurity** to **False**<sup>12</sup>. Integrated security allows an application to login as the currently authenticated user without reentering credentials, but this is only available on Windows.

Once these settings are changed and the configuration file is saved, the console application can be run on the remote machine and connected to the openPDC service.

### Exit Console

The remote console application can be closed by issuing an **Exit** command and pressing enter. Note that unlike when running the openPDC as a terminal application, issuing the **Exit** command from a remote

<sup>10</sup> The configuration files are simple text based XML files and can be edited with most any kind of text editing tool. There is also an included UI based configuration editing tool, **ConfigurationEditor.exe**, which can be run to make changes to the configuration files that is available on both Windows and POSIX platforms.

<sup>11</sup> The default configuration is setup as IPv4 to support as many possible distributions as possible. If the system running the openPDC supports IPv6, the server and client connections can be configured to use IPv6 by specifying “; interface=:0” in the relevant configuration settings and connection strings. The **interface** setting is used to specify the IP of the network interface controller (NIC) to use for the connection – an IP of zero means that the default NIC should be used for the connection; the format of the interface IP setting determines the IP stack version, i.e., IPv4 or IPv6, to use for the connection.

<sup>12</sup> If the command line option is being used from a Windows machine for specifying the server name, this setting must be changed in the config file first. The **IntegratedSecurity** setting is ignored in POSIX environments.





console session will not terminate the remotely running openPDC application, it will only close the remote console session.

### Testing the Default Configuration

When the openPDC is installed and running with its default configuration it includes a sample PMU (repeating file based input), an IEEE C37.118 output stream and a Gateway Exchange Protocol data publisher. The openPDC can be verified to be operating as expected by exercising its outputs.

#### IEEE C37.118 Output Stream

The IEEE C37.118 output stream defined in the default configuration for a POSIX based openPDC installation can be easily tested by running the [PMU Connection Tester](#)<sup>13</sup>. Also, if there is another PDC or application available that can interpret IEEE C37.118, then that system can be connected to the openPDC output stream as well.

The default configuration's output stream will be listening on TCP, IPv4, on port 8900. For simplicity in testing, the output stream is configured with a TCP channel only, so both commands and data will be handled through the same channel.

#### Gateway Exchange Protocol

The Gateway Exchange Protocol, or GEP, is an open source, high-speed, measurement-based publish/subscribe transport protocol used for exchanging time-series data and automatically synchronizing meta-data between two applications. The protocol supports sending real-time and historical data at full or down-sampled resolutions. When sending historical data the replay speed can be controlled dynamically to provide data as quickly as possible, e.g., a data download, or slowed for visualization streaming.

Included with the openPDC POSIX installation is a program that can verify that the GEP publication server is working properly called the GEP Subscription Tester. This application will trend received data from the openPDC in real-time and replay historical data if the local openPDC archive is enabled (see section *Enabling the openHistorian*). The GEP Subscription Tester can be run on the same machine as the openPDC as long as the POSIX environment has a UI<sup>14</sup>.

The GEP Subscription Tester application can be found in the ***/opt/openPDC/GEPTester*** folder.

The GEP Subscription Tester can also be run from another machine and connected to the openPDC remotely. The GEP Subscription Tester can run on Linux, Windows, Mac OS X and Android devices. See the following link to download the tool for use on other platforms:

<http://openpdc.codeplex.com/wikipage?title=GEP%20Subscription%20Tester>

---

<sup>13</sup> Currently the PMU Connection Tester application is still only available from a Windows machine.

<sup>14</sup> The GEP Subscription Tester is a visualization application built using the [Unity 3D](#) gaming platform. This application will only run properly from within an actual UI environment; attempting to use the application from VNC or other virtualized UI environments may not work.



## Configuring the openPDC

Configuration that is rarely changed as well as more advanced settings are defined in the XML based **openPDC.exe.config** file, this type of static configuration is typically only updated when the openPDC service restarts. For more dynamic configuration, e.g., the kind of configuration that can change at run-time but still needs to persist between restarts, is stored in a database.

The database associated with the openPDC contains all of its needed dynamic configuration. More specifically, this database will contain the devices the openPDC will try to connect to, measurements associated with each device, defined protocol-specific output streams, historian outputs used to archive data, the complete role-based security context and other relevant configuration data.

To safely make openPDC database configuration changes it is recommended that all updates be made using the openPDC Manager application, however, when this is not an option the configuration can also be updated manually. The following sections detail the options for updating the openPDC database configuration.

### Manually Updating Configuration using SQL

One way to reconfigure the openPDC is to modify the associated database configuration manually, such as through using Structured Query Language (SQL). Doing this will require some detailed knowledge of the openPDC schema and how the data is used to operate the openPDC.

Some tables in the openPDC schema have an obvious purpose. For example, the **Device** table controls the devices, along with needed connection information, that the openPDC uses to receive data. Other tables, such as the **ConfigurationEntity**, will need further clarification before being safely modified.

The following link contains a detailed schema reference that can be used to assist with making manual updates to the openPDC configuration database using SQL:

<https://openpdc.codeplex.com/wikipage?title=Manual%20Configuration>

For more information on using SQL, see: <http://www.w3schools.com/sql/>.

Note that changes made manually to the configuration database will only get applied to the running openPDC service when the configuration is reloaded, i.e., when the **reloadconfig** command is invoked or when the openPDC service is restarted.

### Updating Configuration Using the openPDC Manager

The openPDC Manager application can be used to simplify configuration of the openPDC. Although this is the ideal choice for modifying the openPDC configuration, currently the openPDC Manager is written as a Windows Presentation Foundation (WPF) application and is not able to execute using Mono<sup>15</sup>; consequently, the manager application will only run on Windows platforms. However, an instance of the openPDC Manager running on a Windows machine can be used to remotely configure and monitor a POSIX based openPDC service.

---

<sup>15</sup> Mono does not completely support WPF. Future versions of the openPDC will use a more portable GUI application development platform such that the same tool can be used on both Windows and POSIX environments.



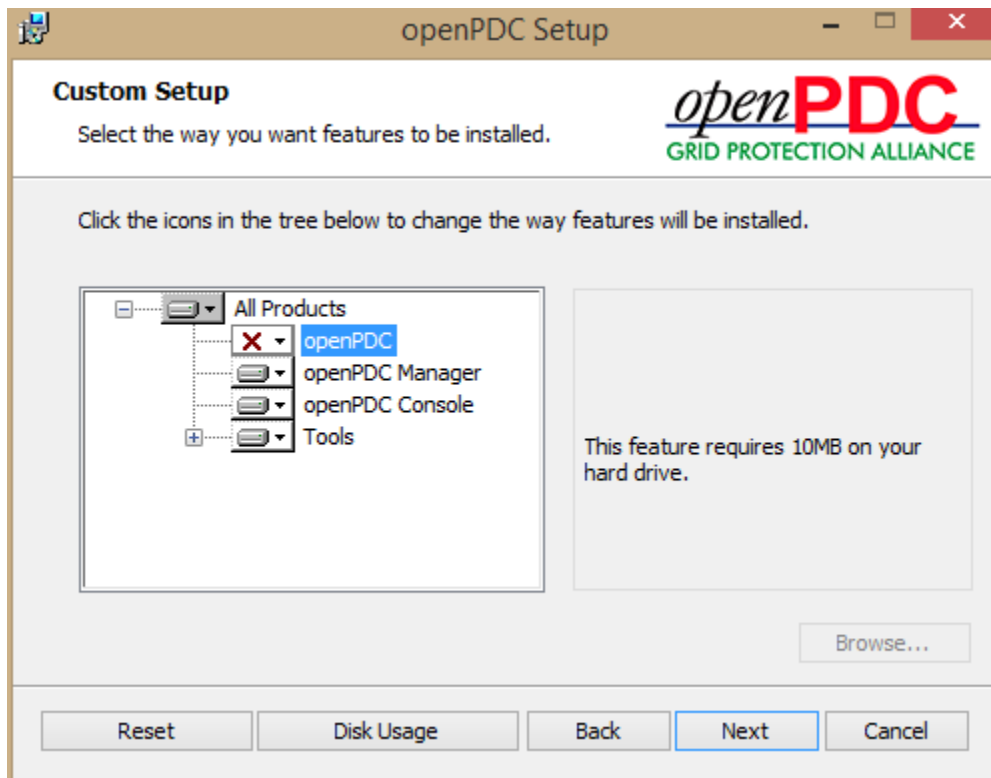
### Installing the openPDC Manager

The openPDC Manager can be installed using the default Windows installer for the openPDC. The latest openPDC Windows installer can be downloaded from the following link:

<https://openpdc.codeplex.com/releases>

From the openPDC release downloads page, choose the recommended **Synchrophasor.Installs.zip** option for download. Once downloaded, unzip the contents of the file into their own folder and run **Setup.exe**.

The openPDC service does not need to be locally installed on the Windows machine, the installation can be reduced to only the openPDC Manager and associated tools. To skip installing the openPDC service on Windows, when the **Custom Setup** step in the installer is displayed, click the “*Entire feature will be unavailable*” drop-down option from the available selections next to the **openPDC** item under the **All Products** listing, for example:



After the installation is complete, the **Configuration Setup Utility (CSU)** will run. For now, click **Cancel** on the CSU main screen to exit the configuration application.

The CSU can be run later at any time to complete the configuration and can even be used to create a new database that both the openPDC Manager running on Windows and the openPDC service running on the POSIX environment can share.



### Using the Default SQLite Configuration

To simplify the setup process, the openPDC POSIX installation comes preconfigured with a SQLite database configuration file<sup>16</sup>. SQLite was selected because the database engine is lightweight, fast, self-contained and requires no additional installation components.

For production-grade deployments of the openPDC, a multi-user Relational Database Management System (RDBMS), e.g., SQL Server, Oracle or MySQL is recommended since these databases generally have better security options and support high concurrency while maintaining data integrity. Additionally since these database systems can be remotely accessed and updated simultaneously from a Windows based openPDC Manager and a POSIX based openPDC service<sup>17</sup>, configuration steps can be greatly simplified. See the *Using Other Databases* section for more information.

Even so, SQLite can continue to be used and will be an ideal choice under these types of conditions:

- (1) The openPDC is being deployed on a device where the configuration will rarely change
- (2) The openPDC is being installed in an environment where only one user will manage configuration
- (3) The openPDC is being positioned as an embedded service on existing hardware (e.g., a DFR)
- (4) The openPDC is being used in a research and development mode

To continue using SQLite as the desired database option and use the openPDC Manager to maintain the configuration, there are basically two ways to get access to the SQLite database file: (1) copy the SQLite configuration database file to Windows, update the configuration using the openPDC Manager, then copy the file back to the POSIX system, or (2) allow the Windows machine to directly access the SQLite configuration database file through a file share mapped to the POSIX machine. These two SQLite configuration options are detailed below<sup>18</sup>.

#### Copy SQLite Configuration for Local Update

Copying the SQLite configuration database file to a Windows machine for local editing using the openPDC Manager is the simpler of the two options for updating the default POSIX configuration. Use the following high-level steps to update a locally copied SQLite configuration on Windows:

- (1) Copy the SQLite database file to Windows machine (e.g., through FTP or SSH file transfer)
- (2) Configure the openPDC Manager to use the local SQLite database file by running the Configuration Setup Utility and using the following options:
  - a. Select “I want to use an existing configuration”
  - b. Select “Database” for configuration type
  - c. Select “I want to use an existing database as-is”
  - d. Select “SQLite” as the database type to setup

---

<sup>16</sup> After a default installation of the openPDC, the SQLite configuration database is stored as a file, **openPDC.db**, in the `/opt/openPDC/Configuration/` folder. If the openPDC destination folder was changed during the installation using the **-d** option with the **install-openPDC.sh** script, the new installation path will need to be used in the command instructions that follow.

<sup>17</sup> For example, connecting to a RDBMS via a socket versus direct file access as is required with SQLite.

<sup>18</sup> Both of the outlined SQLite configuration options assume that security has not already been fully enabled on the openPDC. If security is already enabled, make sure a user for both environments already exists so that the configuration can be readily accessed from the openPDC Manager. See the *Authenticating Cross Platform Users* section for more details.



- e. Select the path to the SQLite database copied locally from the POSIX machine
- f. Click **Next** through all remaining steps (no changes) and openPDC Manager will be launched and connected to local SQLite database
- (3) Update the openPDC configuration as desired using the openPDC Manager
- (4) Exit the openPDC Manager application when configuration updates have been completed
- (5) Copy the SQLite database back to the POSIX machine, overwriting the existing file
- (6) Run the openPDC Remote Console, once connected, execute the **ReloadConfig** command

### Use SQLite Configuration via File Share

In order for an instance of the openPDC Manager running on Windows to directly update the SQLite database configuration file that exists in a POSIX environment, a file share will need to be established that is accessible from the Windows machine. Note that once the desired openPDC configuration has been completed, the file share should be disabled.

To enable a share on a POSIX system, the **net usershare** command can be used, for example:

```
sudo net usershare add openPDC /opt/openPDC "openPDC" everyone:F guest_ok=y
```

Note that the **net** command requires that Samba already be installed, for example, on Ubuntu the following command will install Samba:

```
sudo apt-get install samba samba-common-bin
```

For more information on installing Samba on other platforms, see the following:

<https://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/install.html#id2551914>

To make sure the Windows system can properly access the Samba share, the rights for the folder containing the SQLite database file should be updated to allow reads and writes from external users. As with the file share, the original folder rights should be restored after the desired openPDC configuration has been completed.

To update the folder rights, run the following command:

```
chmod -vR ugo+rw /opt/openPDC/ConfigurationCache
```

Once the share exists, a Windows network drive must be mapped to the folder so that the openPDC Manager will be able to successfully access the SQLite configuration database.

*TODO: Write steps to map Windows network drive then steps to connect manager via CSU. Note that the SQLite browser application required a mapped drive to access database file on Linux share. Need to see if openPDC Manager requires this since a UNC path setting would be simpler.*

Map a drive instructions: <http://windows.microsoft.com/en-us/windows/create-shortcut-map-network-drive#1TC=windows-7>



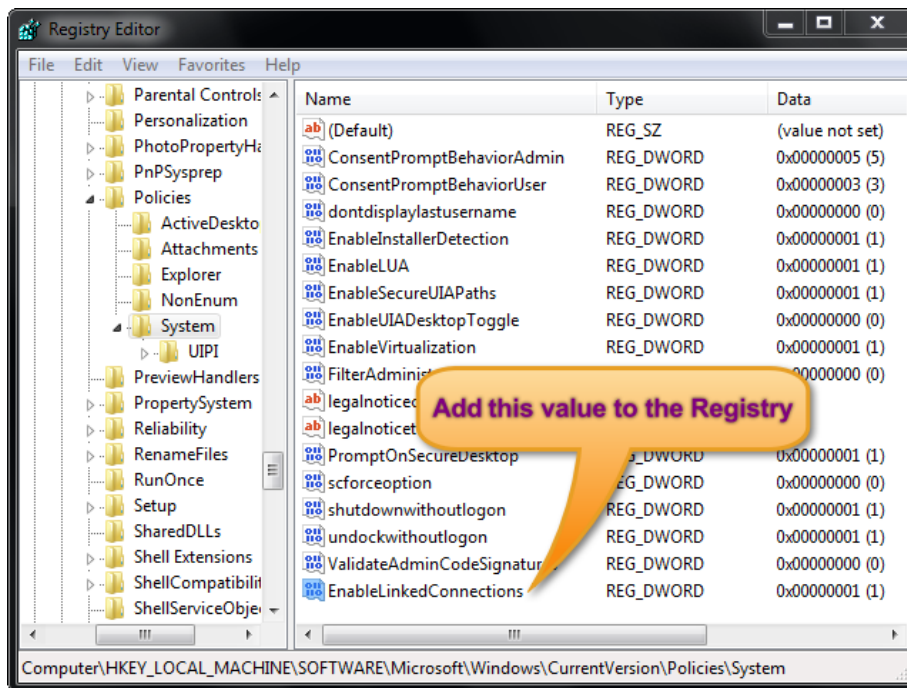
### Troubleshooting Mapped Network Drive

On newer versions of Windows, applications running as an elevated user with administrator privileges may require a registry update in order for the elevated user to see a mapped network drive. If the network drive is not visible after mapping the network drive, use the following steps to make sure mapped network drives on Windows are available to both standard and elevated accounts:

- (1) Launch the Windows Registry Editor application, i.e., **regedit.exe**, and locate the following key in the hierarchy:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System

- (2) Right click on the **System** key folder and create a new “DWORD (32-bit) value” entry named **EnableLinkedConnections** with value of **1**:



**WARNING:** Be careful with the Registry Editor tool, this application can render a system inoperable when modifying settings inappropriately.

- (3) After the **EnableLinkedConnections** value has been added to the Windows Registry, the computer must be restarted. After the system has rebooted, any mapped network drives created should now be available to elevated applications.

### Using Other Databases

Several database types are natively supported by the openPDC, including: SQL Server, MySQL and Oracle. Database configurations can be installed remotely or locally (as supported) with the openPDC service configured to connect to the database for its primary configuration. This can often optimize and simplify interaction with the openPDC Manager since direct database file access will not be required.

The simplest way to setup a new database configuration for an openPDC instance is to run the Configuration Setup Utility that is installed along with the openPDC Manager. This utility is currently only



available on a Windows environment<sup>19</sup>, however, it can create a new configuration for a database system running in a POSIX environment. If the Configuration Setup Utility is being used to convert from one database to another, the **DatabaseMigrationUtility** will be run at the end of the process to migrate the data in the old database to the new one<sup>20</sup>.

Without access to the Configuration Setup Utility, a new configuration database can still be manually created. SQL schema scripts for all supported database types are installed with the openPDC and can be found in the **/opt/openPDC/Database Scripts** folder.

Once a new database has been created, the **openPDC.exe.config** file **systemSettings** category properties called **ConnectionString** and **DataProviderString** have to be updated<sup>21</sup> with connection information for the new database. If the Configuration Setup Utility was used to create the new database, the **systemSettings** category property values applied to the **openPDCManager.exe.config** file by the CSU can often be copied and used as-is<sup>22</sup> into the **openPDC.exe.config** file.

See the following links for more information:

[http://openpdc.codeplex.com/wikipage?title=Getting%20Started#set\\_up\\_database](http://openpdc.codeplex.com/wikipage?title=Getting%20Started#set_up_database)

<http://www.connectionstrings.com/>

### Enabling the openHistorian

The openHistorian 1.0 built into the openPDC also works on POSIX platforms. From a fresh install of the openPDC<sup>23</sup>, download an updated **openPDC.exe.config** and **openPDC.db** configuration database that have the historian enabled by default:

<http://www.gridprotectionalliance.org/NightlyBuilds/openPDC/Scripts/POSIXConfigHistorian.zip>

*TODO: This path needs to be updated to “Products” folder instead of nightly builds – and zip file has yet to be posted.*

To deploy, unzip the files from the download then, with the openPDC is stopped, copy the **openPDC.exe.config** into the **/opt/openPDC** folder and copy the **openPDC.db** SQLite configuration database into the **/opt/openPDC/ConfigurationCache** folder – overwriting the existing files. Restart the openPDC and the local historian will now be archiving data. If the POSIX distribution running the openPDC service also has a UI, the **HistorianPlaybackUtility.exe** found in the openPDC installation folder can be executed to extract data from the archive.

---

<sup>19</sup> Like the openPDC Manager, the Configuration Setup Utility is also currently a WPF based application. This application will be updated in future builds so it will run on both Windows and POSIX platforms.

<sup>20</sup> The Database Migration Utility is Windows Forms based application that is installed with the openPDC service. This application will already run in a POSIX environment with UI support, as well as Windows.

<sup>21</sup> Changes to the **openPDC.exe.config** file should only be made while the openPDC service is not running.

<sup>22</sup> One exception is the **DataProviderString** for SQLite. Under POSIX systems, the SQLite data provider string is: `AssemblyName={Mono.Data.Sqlite, Version=4.0.0.0, Culture=neutral, PublicKeyToken=0738eb9f132ed756}; ConnectionType=Mono.Data.Sqlite.SQLiteConnection; AdapterType=Mono.Data.Sqlite.SQLiteDataAdapter`

<sup>23</sup> If the openPDC system is already installed and configured and the current configuration needs to be preserved, the historian must be enabled manually in the configuration or by using the openPDC Manager.





## Completing Security Configuration

Although security is fully enabled by default when installing the openPDC on Windows platforms, completely enabling security on POSIX platforms, e.g., Linux and Mac OS X, is a multi-step process that includes building the Grid Solutions Framework (GSF) shared POSIX library that requires distribution specific libraries<sup>24</sup>. As a result, the default configuration that gets installed on a POSIX platform has some security features initially turned off to simplify deployment. The following sections describe how to fully enable all the security features of the openPDC when running in a POSIX environment. The example steps shown below are for Ubuntu, but this may be different for any given distribution<sup>25</sup>:

- 1) Get the GNU Compiler Collection (gcc) for your distribution, e.g., for Ubuntu:

```
sudo apt-get install build-essential
```

- 2) Get the Pluggable Authentication Module (PAM) libraries for your distribution, e.g., for Ubuntu:

```
sudo apt-get install libpam0g-dev
```

- 3) With the openPDC stopped, execute the security setup script<sup>26</sup>:

```
sudo bash enable-security.sh
```

The **enable-security.sh** script will compile and deploy<sup>27</sup> the shared GSF POSIX library and update the openPDC configuration to require secured remote interactions, i.e., commands can only be issued by authenticated users meeting a minimum role-based permission (e.g., the **Administrator** role).

For reference, the script is modifying the **openPDC.exe.config** file and changing the following setting:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <ategorizedSettings>
    <serviceHelper>
      <add name="SecureRemoteInteractions" value="True" />
    </serviceHelper>
  </ategorizedSettings>
</configuration>
```

Once the GSF POSIX library has been compiled and the security updates have been applied, restart the openPDC. At this point the openPDC will be essentially locked-out; i.e., until a user with a valid role assignment is added to the system, no remote command access to the openPDC will be allowed without

---

<sup>24</sup> In order to enable user authentication and validation on a POSIX platform, the GSF POSIX shared object must be compiled locally using libraries, such as PAM, specific to the distribution running the openPDC.

<sup>25</sup> For Mac OS X, steps one and two can usually be replaced simply by installing the command line developer tools, such as those that can be installed from the Xcode component downloads.

<sup>26</sup> This script is downloaded as part of the installation process and can be found in the original installation folder along with **install-openPDC.sh**. The script depends on the source code being preserved during the installation, i.e., make sure to use the **-p** parameter when installing the openPDC.

<sup>27</sup> Once compiled, the shared object library, GSF.POSIX.so, will be deployed into the openPDC installation folder.





user verification<sup>28</sup>. Once a user is authenticated by the openPDC with a valid role assignment, all commands issued to through remote sessions require a minimum role-based permission for the issued command. For more information on role-based security in the openPDC, see the following documentation:

<http://openpdc.codeplex.com/wikipage?title=Remote%20Console%20Security>

### Managing User Accounts

The openPDC supports two kinds of users: (1) system authenticated users and, (2) database authenticated users. As their name suggests, system authenticated users will be authenticated by the local operating system according to configured authentication policies, e.g., the PAM configuration. This can be local accounts or domain accounts, if enabled. Database authenticated users will be authenticated by matching a user name and a hash of the password as defined in the openPDC configuration database. Generally, system authenticated users are the ideal choice since the user credentials and account information is maintained by the operating system.

Groups, both for system and database, are also supported. Groups allow easy management of role assignments to any set of users that exist within a group. Note that explicitly defined role assignments for a user will always take precedence over implicitly acquired role assignments that are derived from a group that contains the user as a member.

### Adding New Users

*TODO: Complete section – also include “add-user.sh” script instructions.*

<Need to mention installing sqlite3>

### Authenticating Cross Platform Users

The openPDC can be configured to allow authentication from both POSIX and Windows environments simultaneously. Cross platform authentication comes into play when using the openPDC Manager running on Windows to manage the configuration of an openPDC service running in a POSIX environment.

Since the openPDC Manager application can directly manage a configuration database even without access to a running openPDC service, it has to independently authenticate a user. Also, since the openPDC Manager currently only runs on Windows, its user authentication will always be Windows based. However, when connected to a remote openPDC, the openPDC Manager also has to authenticate the user with the remote openPDC service, which can be running in a POSIX environment.

To allow cross platform authentication, a user entry must exist in the openPDC configuration database for each platform for which a user needs to be authenticated. To prevent user configuration duplication, system users and groups added to the openPDC configuration are converted to a corresponding system identifier (SID) as provided by the operating system. Duplication is prevented because different

---

<sup>28</sup> Note that the openPDC primary functions, e.g., PMU data acquisition, IEEE C37.118 output streams and GEP data publishers, will continue to function as normal as configured, but no runtime commands or configuration changes will be accepted without proper user authentication and role validation.



operating systems provide different system identifiers<sup>29</sup>, allowing the same user to be added to the security configuration for different platforms.

### *Cross Platform Domain Users*

When both the POSIX and Windows environments are configured to authenticate against the same domain, a domain user or group can be added for both environments effectively.

When adding a domain user to the database configuration the domain name prefix must be specified. For example, if the domain name is “GPA” and the user name is “openPDCUser”, the username to be added to the database configuration should look like “GPA\openPDCUser”<sup>30</sup>.

When a system user is added to the database as a domain prefixed username and is subsequently recognized as a system user during the next openPDC service configuration load, e.g., issuing the **reloadconfig** command, the system user will be changed to an operating system specific SID in the database. When adding a recognized domain user from the openPDC Manager, it automatically writes the Windows SID to the database. Accordingly, the domain user will need to be added from the openPDC Manager for proper authentication on Windows and added to the configuration database from within the POSIX environment for proper authentication by the openPDC service running there.

### *Cross Platform Local Users*

The only current way to manage a cross platform local user is to create a user with the same name and password on both the POSIX and Windows environments. However, this does not have to be a local system account on both systems – one option is use a database user in one environment with the same credentials as the local system user in the other environment<sup>31</sup>.

### Disabling Pass-Through Authentication

There is no functionality in a POSIX environment to impersonate a user’s security context, so any Windows based openPDC pass through authentication and integrated security need to be disabled. In practice this means credentials will be required to login to the openPDC Manager on Windows when connecting to an openPDC running on a POSIX platform, e.g., Linux or Mac OS X. As a result, if the Windows based openPDC Manager is being used to maintain a POSIX openPDC configuration, the openPDC Manager configuration must be modified to require login credentials<sup>32</sup>.

---

<sup>29</sup> Helping with uniqueness, POSIX platform user and group SIDs for the openPDC will also be prefixed with “user:” and “group:” respectively. This is needed to create a unique SID that spans both user and group namespaces because in POSIX environments users and groups are managed separately and their IDs often overlap.

<sup>30</sup> Use of the backslash delimiter for the openPDC configuration usernames is required even if the PAM configuration on the POSIX configuration commonly uses another symbol to delimit the domain and user name. Consequently, if the openPDC configuration database being used is sensitive to characters escaped with a backslash, it may be necessary to format the username like “GPA\\openPDCUser”.

<sup>31</sup> When adding a database user to the openPDC configuration database, only the password’s hash will be stored, not the actual password.

<sup>32</sup> Technically with a properly defined Windows account in the configuration database, the openPDC Manager can still login with pass-through authentication. However, in order to also successfully connect to the openPDC service running in a POSIX environment and issue commands, valid user credentials must be collected.



To modify the openPDC Manager configuration to always request login credentials, stop the openPDC Manager, edit the **openPDCManager.exe.config** file and change the following setting:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <userSettings>
    <openPDCManager.Properties.Settings>
      <setting name="ForceLoginDisplay" serializeAs="String">
        <value>True</value>
      </setting>
    </openPDCManager.Properties.Settings>
  </userSettings>
</configuration>
```

### Updating Transport Layer Security

The following documentation describes how to manually update Transport Layer Security (TLS) settings. The openPDC TLS configuration is automatically enabled in the installation script along with a newly created self-signed certificate, as a result, steps to update TLS are generally not required. However, this section is provided in case the TLS configuration needs to be updated or customized.

For high level information related to using TLS/SSL with Mono, be sure to review the following:

<http://www.mono-project.com/docs/faq/security/>

### Manually Creating a Certificate

Using the **MonoGenCert** tool that gets deployed with the openPDC will be the simplest way to generate a new certificate, for example, change directory into the openPDC installation folder (typically **/opt/openPDC**) and execute the following:

```
sudo mono MonoGenCert.exe openPDC
```

Internally this command calls **makecert** with the proper parameters to create all the needed keys and certificate files. The tool also automatically generates a list of all known common names for the machine.

For more control of the certificate creation process, the **makecert** command can be used directly, for example:

```
makecert -r -n "CN=openPDC,CN=linux.domain.com,CN=192.168.1.149,CN=..." -p12 openPDC.p12  
"" -sv openPDC.pvk openPDC.cer
```

When creating certificates using the **makecert** command, note carefully the list of common names after the **-n** flag. If the list does not contain an entry matching that which was used to connect to this server, then the client configurations, e.g., console and manager applications, will have to add **RemoteCertificateNameMismatch** to the list of **ValidPolicyErrors**. Note that the only exception is localhost, which the local client applications will trust regardless.

If the **makecert** command completes successfully, the result will be two new files. The first file is **openPDC.p12** which contains the public key, certificate information and the private key – this file is



required by Mono in order to properly secure network traffic. The second file is **openPDC.pvk** which is the standalone private key file<sup>33</sup>. If needed, call the **makecert** command again without the **-p12** parameter to create a public key certificate file, i.e., **openPDC.cer**<sup>34</sup>.

To protect generated private keys, make sure to restrict access to the files<sup>35</sup>:

```
chmod 600 openPDC.p12 openPDC.pvk
```

### Using an Existing Certificate

Existing certificates, such as one obtained from a public certificate authority, can be used by updating the needed **openPDC.exe.config** file settings:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <categorizedSettings>
    <remotingServer>
      <add name="LocalCertificate" value="openPDC.p12" />
    </remotingServer>
  </categorizedSettings>
</configuration>
```

As always, when making modifications to the openPDC configuration, stop the openPDC, make the needed changes, and then restart the openPDC. Additionally, as mentioned in the *Manually Creating a Certificate* section, always make sure to protect the private key files.

### Changing the TLS Version

As of version 3.12 of Mono, only TLS 1.0 and older protocols, e.g., SSL versions 1.0 to 3.0, are currently supported<sup>36</sup>. The default configuration of the openPDC installation enables the highest level security available for the environment. To change the supported transport protocols, stop the openPDC, edit the **openPDC.exe.config** file and update the following setting:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <categorizedSettings>
    <remotingServer>
      <add name="EnabledSslProtocols" value="Tls" />
    </remotingServer>
  </categorizedSettings>
</configuration>
```

As newer versions of Mono are released that enable updated versions of the TLS protocols, just update the setting with the latest protocol version, e.g., **Tls12** – see table below. Note that client application

---

<sup>33</sup> If the private key file **openPDC.pvk** already exists before the call to **makecert**, a new file will not be created – the tool will instead use the existing private key.

<sup>34</sup> Using the **MonoGenCert** tool automatically creates all three files.

<sup>35</sup> This step happens automatically when using the installation script.

<sup>36</sup> With the recent open sourcing of Microsoft's .NET framework, Mono is making improvements on many fronts, including security. As of the writing of this document, [TLS1.2 has been enabled in the latest Mono source code](#) but has not yet been made available in a release build. When Mono gets a new release that contains updates that the openPDC can use for improvements related to security, e.g., TLS 1.2, these updates will be tested with the openPDC so that improved security features can be enabled as soon as they are available.



configuration settings, e.g., the console and manager, should be updated to match the minimum specified server protocol in order to properly connect.

Multiple security protocols can be specified separated by a comma, the possible options are as follows:

Protocol	Description
<b>None</b>	No SSL protocol is specified. Cannot be combined with other protocols.
<b>Ssl2</b>	Specifies the SSL 2.0 protocol. SSL 2.0 has been superseded by the TLS protocol and is provided for backward compatibility only.
<b>Ssl3</b>	Specifies the SSL 3.0 protocol. SSL 3.0 has been superseded by the TLS protocol and is provided for backward compatibility only.
<b>Tls</b>	Specifies the TLS 1.0 security protocol. The TLS protocol is defined in IETF RFC 2246.
<b>Tls11</b>	Specifies the TLS 1.1 security protocol. The TLS protocol is defined in IETF RFC 4346.
<b>Tls12</b>	Specifies the TLS 1.2 security protocol. The TLS protocol is defined in IETF RFC 5246.

### Running the openPDC with Elevated Rights

In POSIX compatible applications, security for an application starts high then is reduced to a user's permission level upon access; the openPDC is no exception. To make sure the openPDC has access to needed security information for user authentication and validation, it must be started with elevated privileges<sup>37</sup>.

When running the openPDC as a daemon, the service application registration script already makes sure the openPDC is running with needed elevated privileges such that security functions can be accessed as needed.

### Securing the Configuration Database

The default configuration deployed during an openPDC POSIX installation includes a simple SQLite database for its configuration. To continue using the SQLite database, any Samba share enabled in prior steps for using SQLite with the openPDC Manager, e.g., in the *Use SQLite Configuration via File Share* section, should be disabled. Any Samba share should only be enabled while configuration changes are being made, then subsequently disabled. Additionally, the share should be reconfigured to require security and the access rights of the **/opt/openPDC/ConfigurationCache** folder should be reduced to a specific set of users.

---

<sup>37</sup> If the openPDC was installed using **sudo**, folder rights will already require the openPDC to run with elevated privileges; however, elevated privileges will still be required by the openPDC when security is fully enabled even if folder rights are changed not to require root access. Without the needed run-time privileges with security fully enabled, the openPDC will not be able to validate user credentials and remote command access into the openPDC will be locked-out.



## Deploying the openPDC on POSIX Platforms

For optimal database configuration security, migrating to another type of RDBMS and using its security mechanisms is highly recommended. SQL Server<sup>38</sup>, MySQL and Oracle can all be used with the openPDC. See the *Using Other Databases* section for more information.

Enabling database security is specific to the RDBMS type being used. See the database specific SQL schema scripts installed with the openPDC in the ***/opt/openPDC/Database Scripts*** folder for comments on enabling security and/or adding database users.

Where possible for databases that support it, some security and account creation will be handled during new database creation when using the Configuration Setup Utility. However, always consult the vendor specific RDBMS security documentation to make sure security is being properly setup according to recommended practices such that both the openPDC service and the openPDC Manager are appropriately authenticating with the database.

---

<sup>38</sup> An openPDC running in a POSIX environment can connect to SQL Server running on Windows.



## Running on a Raspberry Pi

At the time of writing there were no common distributions for the Raspberry Pi that contained Mono at version 3.12; therefore, Mono must be compiled before openPDC installation. In our tests, it took about 8 hours to completely build the needed Mono components on an original Raspberry Pi system and less than 3 hours on a multicore Raspberry Pi 2 system.

To avoid needing to compile Mono and speed up the installation process, GPA has posted an image for download with the needed version of Mono and the openPDC preinstalled for running on a Raspberry Pi and Pi 2 with the Raspbian OS. See the following link for instructions:

<https://openpdc.codeplex.com/wikipage?title=Running%20openPDC%20on%20a%20Raspberry%20Pi>

The posted image comes with the default openPDC configuration. Additionally, the openPDC is set to automatically run at startup as a daemon and has security fully enabled.

For best openPDC performance, the Raspberry Pi 2 is recommended. The new Raspberry Pi 2 Model B has 4 cores, 1 GB of RAM and better CPU performance all of which provide a very practical and performant micro-environment for running the openPDC.

The openPDC also runs on the original Raspberry Pi. For optimal performance on this single core system it is recommended that the configuration of the openPDC on the Raspberry Pi be reduced to its primary tasks.