Corkscrew Content Management

Developer’s Guide

Copyright © 2016. Aquarius Operating Systems, India. All Rights Reserved.

Aquarius Operating Systems and Corkscrew are registered trademarks. Misuse and unauthorized usage of these names or branding will attract penal action and prosecution to the extent of applicable laws.

This is a guide for developers seeking to leverage the Corkscrew Content Management System SDK to create client, server and other applications. The information in this guide is provided on an as-is basis. Aquarius Operating Systems makes or implies no warranties or guarantees or assurances of any kind or assume any liabilities for damages arising out of usage of the content in this guide, including code examples.

Code examples provided in this guide may compile and function normally. But these are provided solely for illustrative purposes.

# Getting Started

Welcome to the developer’s guide for Corkscrew Content Management System. We shall call this Corkscrew for short from here on. This guide describes the functionality provided by Corkscrew to developers to create enterprise-class content management software. We also describe in detail and illustrate with examples the different features that Corkscrew supports.

Specifically, you will learn about:

1. Kicking things off with Corkscrew
2. Understanding the different capabilities and functionalities in Corkscrew
3. Creating and managing Corkscrew farms, sites and other objects
4. Creating, managing and working with workflows in Corkscrew
5. Setting up and fine tuning user access control
6. Leveraging the capabilities of Corkscrew in other typical enterprise applications

## About Corkscrew

Corkscrew is built as a framework or SDK. It provides a content management system integrated with a workflow that can be used to automate business process tasks. Corkscrew helps developers and IT administrators create farms, sites and, hierarchical content storage.

## How Corkscrew CMS works

Corkscrew is a two-layer application. It consists of a backing database, where the content being managed is persisted and the application layer where the content managing software executes. The workflow system also runs within the application layer.

### The backend

Any CMS system needs to store the content in some form of a backend. Typically, this backend is a database (like Microsoft SQL Server, MySQL Server, Oracle database or something similar). With advances in content storage and retrieval systems, this backend system can be any method of storing, indexing and quickly retrieving information when required. CMS systems also require storage of metadata and historical (change history) information about the content they are managing.

Corkscrew supports backing the content in practically any content storage mechanism. Out of the box, it comes packaged with support for:

* Microsoft SQL Server
* MySQL Server.

When used with these two platforms, the database scripts also install server side database code, in the form of stored procedures (or routines), triggers and database jobs. This database code encapsulates some non-critical business logic – like data validation and cascading update and delete operations. This database code does not contain code to perform access control or locking related checks and operations – that is contained in the SDK library and is done in the application layer.

NOTE: The implication of the separation of business logic between database and SDK layers means that one cannot write code that runs purely on the database side without causing serious impact to the system. However, this is NOT a supported scenario – developers or administrators who insert such code into their Corkscrew databases should take care that such code does not cause irreparable damage to the data.

However, it provides extensibility to let other developers create providers for any other system they prefer (like Json files or the operating system’s file system). This makes it versatile, scalable and powerful as it is not limited to a platform anymore.

### SDK Library

The object model and business logic of the Corkscrew system are contained in the SDK library. All required validation, locking, access control and other related functions required for an operation are performed automatically. Developers leveraging the SDK in their implementations only require to write minimal code to perform any operation.

### Windows Service

Corkscrew packages a Windows Service. This can be installed on any system that has access to the backend (database). If the backend system is a Windows system, then it can be installed on that system as well. This service exists purely to execute the workflows contained in the Corkscrew system.

If the Workflow Service is stopped, it does not cause any other impact to the system, other than that any workflows will fail to run. However, any workflows that were started before the service was stopped may leave the system in an undesirable state. Therefore, workflows must be examined for such negative impact before the service is stopped.

## What developers can do with Corkscrew SDK

The Corkscrew system does not mandate a particular method of access to its systems. For example, it does not enforce access through a bundled web interface or through a particular tool. The SDK provides free reign to the developer to create their own front-ends and other facades to encapsulate and present the underlying functionality in any manner they please. Developers can further enhance or “dumb down” (by placing custom restrictions on what their users can do) Corkscrew through the code they write. This presents enormous possibilities in how, where and why Corkscrew can be used by customers.

## Compatibility with Mobile and Cloud

Corkscrew’s SDK layer is coded in .NET Framework 4.6.1. Hence, applications written with it can be hosted in any container that is .NET Framework capable. This includes desktops, server systems, laptops, mobile devices and cloud infrastructure and more. On native Windows systems, Corkscrew SDK can be targeted through Microsoft’s .NET Framework layer. On non-Windows systems (such as Linux), the SDK can be executed through the MONO .NET framework.

Thus, Corkscrew itself is compatible with any platform or system that can either host or communicate with a system that can host either the .NET or MONO platform.

## Setting up a Corkscrew Environment

Corkscrew can be setup in a partially automated fashion. But first let us set up the pre-requisite environment.

### Installing pre-requisites

1. The system must be capable of running either:
   1. Microsoft .NET Framework 4.6.1 (typically a Windows system)
   2. Or MONO.NET (typically a Linux system).
2. Ensure you have at least over 1 GB of free RAM memory available on the system after the OS has booted. This means really low end systems like the Raspberry Pi 1 and Raspberry Pi 2 cannot be used – trying to use them will result in “Out Of Memory” errors while executing Corkscrew SDK code.

**Note that on a Linux system, Workflows are not supported at this time.**

Once you have determined to the base system (hardware and OS) you will be using, go ahead and install the following on it:

1. If you are on Windows, install .NET Framework 4.6.1 ( link: <https://www.microsoft.com/en-us/download/details.aspx?id=49981> ). If you are on Linux, install the latest MONO by running this command:

sudo apt-get install mono-runtime

1. Select a suitable database platform and install the requisite software for it. Out of the box, the following database engines are supported:

* On Windows:
  + Microsoft SQL Server 2008 and above
    - “Express” and “Developer” editions can be used, however these editions of some versions of SQL Server cannot run SQL Jobs. This may minimally affect the Corkscrew system.
  + MySQL Server 5.7 and above
* On Linux:
  + MySQL Server 5.7 and above

1. If the .NET/Mono system and the database system are two different machines or VMs, ensure that the firewall ports on the database system allows communication for the selected database engines.

### Installing Corkscrew

Here we see how Corkscrew can be installed to a system. Ensure that you have performed the steps listed in the above Installing Pre-requisites section before you continue below.

(Depending on how you got your copy, some of the files may be present as a folder or in a .Zip file)

1. If you are using a connection string based database system, login to that system using its available GUI or command line tool. Create a user with a password, and give that user Administrator privileges on the database engine (the user must be able to create databases, execute jobs and do all administrative tasks on that database engine instance).
2. From the distribution, copy or extract the files from the “Corkscrew Explorer” folder or zip file to a folder on your system’s hard disk.
3. If you are using a backend other than Microsoft SQL Server or MySQL Server, now is a good time to copy the relevant provider’s DLLs into the Corkscrew Explorer folder.
4. The contents you copied should contain a file named “CorkscrewExplorer.exe.config” – if it does not contain this file, create one with the following minimal configuration:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <configuration>  <startup>  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1"/>  </startup>  <connectionStrings>  <add name="configdb" providerName="System.Data.SqlClient" connectionString="Data Source=(local);Initial Catalog=Corkscrew\_ConfigDB;User Id=INVALIDUSER;Password=INVALIDPASSWORD;Connect Timeout=15;Encrypt=False;TrustServerCertificate=False;Pooling=true;Packet Size=32768;"/>  <add name="sitedb" providerName="System.Data.SqlClient" connectionString="Data Source=$(DBServer);Initial Catalog=$(DBName);User Id=INVALIDUSER;Password=INVALIDPASSWORD;Connect Timeout=15;Encrypt=False;TrustServerCertificate=False;Pooling=true;Packet Size=32768;"/>  </connectionStrings>    </configuration> |

**NOTE: If you have the official distribution of Corkscrew, you do not need to perform any changes here.**

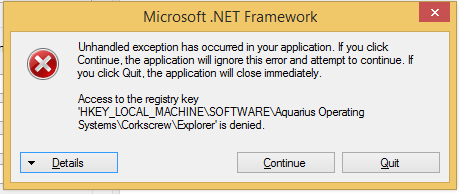
Take a look at the highlighted portions of the “connectionString” value of the “configdb” connection string. The values there for Data Source and Initial Catalog must NOT point to an existing combination of server and database for your environment.

If you are using a database system other than Microsoft SQL Server and MySQL, ensure that the connection strings are in a compatible format.

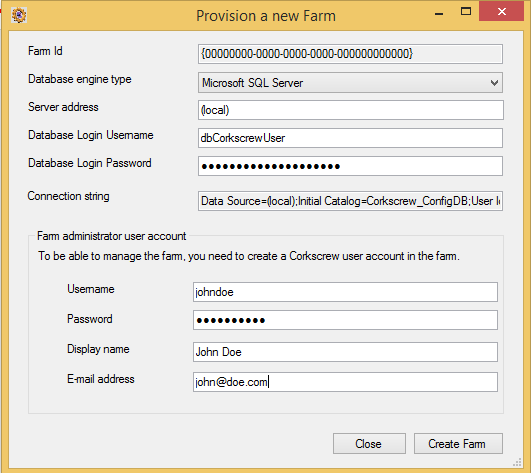
You cannot change the name of the configdb database, it must be called Corkscrew\_ConfigDB. Therefore, you can point the connection string to a system that does not have that database. Once you have made the edit, save and close this file.

1. Double-click on the “CorkscrewExplorer.exe” to launch the UI tool. It will take some time to attempt to connect to the server and database indicated and then you will see a popup to “Provision a New Farm”.

If you see this message box, you may either select the “Quit” button and re-launch the tool in Administrator mode (right-click, Run as Administrator), or Continue with no negative effect – you will simply see a dialog box later on every launch.



Continue to Provision a new farm using the popped up dialog box:



**NOTE: At this point of time, only a limited set of database engines may be supported through “Provision a New Farm” UI of the Corkscrew Explorer tool. If your backend is not supported, use the automated method or run the database installation scripts manually (sorry, no instructions possible!) and then suitably edit the .config file.**

For the “Database Login Username” and “Database Login Password”, use the information for the user account you created in Step 1 of this section.

Click on Create Farm and wait a few moments for the farm to be provisioned. Once successful, you will be notified to copy the connection string and edit the CorkscrewExplorer.exe.config file. Do so now – you should change the value of the “configdb” connection string with the entire connection string as shown in the Provision a New Farm dialog.

1. If you are on a Windows system,
   1. From the distribution folder, find the “Corkscrew Workflow Service” folder or Zip file. Extract its contents to a different folder. This folder needs to be around for a long time, so place it in a carefully chosen location.
   2. In the extracted folder, find the file named “InstallService.cmd” and run it. This will install a Windows service named “Corkscrew Workflow Service”. It will be installed to execute under “Network Service” account and set to “Automatic (Delayed Start)” start up, but not started yet – there are no workflows to execute on a freshly installed system!

Your Corkscrew system is now fully setup.

## Connection Strings

For a database-backed installation, Corkscrew uses connection strings. In the configuration file, Corkscrew SDK expects two connection strings:

<add name="configdb" connectionString="..."/>

<add name="sitedb" connectionString="..."/>

The “configdb” connection is used to connect to the Farm and to perform all farm-level operations, like setting up users and security, deal with sites and workflows and so on. The second “sitedb” connection string is used only to connect to a particular Corkscrew Site. This separation allows the underlying ADO.NET based connectivity layer to pool connections in an efficient way between farm and site level connections.

Do note that the “sitedb” connection has two special tokens embedded in the string. These are NOT to be changed.

* The “$(DBServer)” token is dynamically replaced with the server location.
* The “$(DBName)” token is dynamically replaced with the name of the Site’s content database.

These replacements occur in Corkscrews database interaction layer at the time of setting up the relevant connection.

## Dependency DLLs for Development

Any application wishing to use the rich object model of the Corkscrew SDK should naturally add reference to the “aquariusos.corkscrew.library.dll”. This DLL in turn has dependencies that need to be present (though not installed). That is, those dependencies can be present in the same folder as the SDK DLL file or in any assembly search path on that system. To keep it simply, ensure that all the DLLs in the folder with the SDK DLL file are present in your application’s BIN folder.

# Understanding the Corkscrew Object Hierarchy

In the Corkscrew SDK, most classes are prefixed with “CS”, this stands for “Corkscrew”.

## CSUser or the User

The CSUser class represents a single user account. This class contains only a few basic attributes of a user account – the display name, the username and e-mail address.

The username is used to login to the Corkscrew system. Workflows and other subsystems require participating users to have an e-mail address. The display name is only relevant to user interface systems – it is a friendly name that could be displayed on user interfaces.

The CSUser class contains methods to change the current password using a given password or by an automatically generated password. Statically, the class also contains methods to create a new user, delete the user, login a given account with a username and password or validate a particular user.

This class is backed by the “**Users**” table in the backend.

## CSPermission or the ACL

Access control or ACL in Corkscrew is based on three attributes connected to the target object. The three attributes are:

* Read – user may access and read the data and metadata associated with the object, but not modify or delete it.
* Contribute – Includes all access granted by Read, but user may also modify the object and create child objects (if possible). However, user cannot delete the object.
* Full Control – the user may perform any activity against the target object.

These permissions are hierarchical. That is, granting the Contribute access automatically also grants the user Read access. Full Control automatically grants Contribute and hence Read.

Additionally, there is a fourth attribute “Hierarchical Access” (called as “Child Access” in the backend). If this attribute is set, it means that the ACL attributes of Read, Contribute and Full Control are not set on the target object but have been granted because of a specific permission deeper in the hierarchy. Let us clarify this with an example ---

Let’s assume a Uri to an imaginary object as:

/foo/bar/object

Now if a user is denied access to either “foo” or “bar”, that user cannot access “object”. Therefore, to properly grant the user access to “object”, the user must be given sufficient access to “foo” and “bar” as well. This is called hierarchical or child access in the Corkscrew system.

Hierarchical access is not granted specifically. This attribute is set automatically when hierarchical access is required.

The CSPermission class contains the method TestAccess to retrieve the current ACL for a given object combination. The caller may then modify the attributes and save it back to change the access.

This class is backed by the “**Permissions**” table in the backend.

The CSUser and CSPermission classes reside in the **aquariusos.corkscrew.library.security** namespace. All the below classes reside in the **aquariusos.corkscrew.library.objects** namespace.

## CSFarm or the “Farm”

At the root of the object hierarchy is CSFarm, we call this the Corkscrew Farm. This is a “virtual” object, as it does not have any direct representation in the backend. The primary members of the CSFarm class are collections of various members of a Corkscrew Farm.

Using the CSFarm object, the developer can access all Farm level objects:

* Users or only users with administrative access
* Configuration
* Content types or registered MIME type names for different file extensions
* Sites
* The Farm filesystem

The Farm’s file system is a collection of directories and files that are not connected to a particular Site. These objects are globally available throughout the Corkscrew installation. For example, a resource that is required to be available commonly across all the Corkscrew sites in the farm can be placed at a particular “deep” location in the Farm file system. It can then be transparently accessed without the caller having to perform checks at different locations.

CSFarm class also contains methods to create a new Site or User in the farm.

## CSSite or Site

The CSSite object represents a single Site in a Corkscrew Farm. A site (CSSite) is a child object of the farm (CSFarm). Sites are basically hosts of a file system with each site starting off with a root directory with directories and files under it. Corkscrew Sites are not hosts of any other objects other than directories and files – for example, there are no site-scoped users. However, workflows may be associated with and executed in scope of sites.

In the backend, the CSSite is backed by the “Sites” table.

The CSSite object provides accessors for users, content types, etc similar to CSFarm. However, these are simply shortcuts to calling the parent farm’s properties.

**NOTE:** Developers can use the ContentDatabaseServerName and ContentDatabaseName properties of the CSSite class to retrieve the server address for the site’s content database and the name of the database. However, at this time, these properties cannot be set – they are fixed to the server address of the farm’s ConfigDB database and the Corkscrew\_ConfigDB as the database name. In a future version, it will be possible to change the names and point the site to be served out of a different server and database.

In Corkscrew, a Site may or may not be “bound” to a “website”. This is a departure from a traditional CMS system where their site equivalents (that is the container for their file system or content) is a web based entity. When a Corkscrew Site is bound to a website, then the list of DNS names that the site must respond to are configured using the CSSite class’ DNSNames property. Each site may have a number of DNS names bound to it. But a single DNS name can be bound only to a single CSSite entity.

The amount of content that you store into a site’s file system is controlled using quota limits. This is configured in the CSSite’s QuotaBytes property. A value of 0 means quota is disabled, meaning there are no limits enforced by Corkscrew.

Note that the farm’s file system is hosted in a transparent CSSite. This CSSite is called the “ConfigDB Site” or the “Configuration Site”. This special site has the empty Guid as its Id.

## CSFileSystemEntry

This is a base class and is not directly instantiable. It contains properties and methods that are common to the CSFileSystemEntryDirectory and CSFileSystemEntryFile classes described below.

### CSFileSystemEntryDirectory or Directory

This class represents a single directory. It derives from CSFileSystemEntry. It has standard properties of files and directories (like name, extension, etc) as with common file systems. As a directory, it also provides access to child directories and contained files; and provides methods to create them.

### CSFileSystemEntryFile or File

This class represents a single file and derives from the CSFileSystemEntry. It has standard properties of files and directories (like name, extension, etc) as with common file systems. As a file, it has methods to open, close, read and write data to the file.

The CSFileSystemEntryFile class leverages the **CSVirtualFileStream** class to provide Stream I/O access to the file’s content. The CSVirtualFileStream implements a stream over the Stream base class.

## Collections of Sites, Directories and Files

Corkscrew container objects allow retrieval and management of their child elements using collection-based properties. For example, a set of CSSites are children of a CSFarm, and a set of CSFileSystemEntryDirectory and CSFileSystemEntryFile entities are children of a CSSite. For such collection properties, the Corkscrew object model defines a base class called “**CSBaseCollection<T>**”. Classes such as CSSiteCollection, CSFileSystemEntryDirectoryCollection and CSFileSystemEntryFileCollection are derived from CSBaseCollection<T>. These collection classes implement methods to deal with a set of the particular object and allow for add, remove and search functionality.

Commonly, these methods implement the following create, remove or search methods:

* Add (with one or more overloads) – allows creation of a new child element. The new element would be created and persisted in the backend, added to the collection and then returned.

An exception is the Add overload that takes the child element class type as the parameter. This overload will not create a new child element, but simply add the passed in instance to the collection.

* Remove (with one or more overloads) – all the overloads will delete the child element specified and remove it from the collection.
* Clear – Clears the collection of all elements, but does not delete any of the elements from the backend. For example CSSiteCollection.Clear() will not delete all the sites.
* Exists – Searches for an element with the given property and returns if it is found.
* Find – searches for an element with the given property and returns the element if found.

## The CSPath

This class provides functionality equivalent to the System.IO.Path class in the .NET Framework. It has a set of static methods that provide path resolution and path combination functionality. Additionally, the CSPath class provides abilities to deal with the Corkscrew Resource Uri string and resolve it to figure out what it represents.

### The Corkscrew Resource Uri

Corkscrew uses “resource locator” Uri to locate an entity within its hierarchy. The format of this string is:

**corkscrew://[**<guid of CSSite>**]/***path/to/resource/file.ext*

An example of this string would be:

**corkscrew://[**cba61c39-9200-4535-8332-5e2c5d995e78**]/***My Documents/Corkscrew/help.htm*

As you can see, the resource Uri is not bound by the Farm the resource belongs to. However, this is okay since the resources are rooted within the respective Site containers that are separated by Guids. The only collision expected is when resources reside in the Farm’s file system as the Id of all of these sites would be the empty Guid.

The GetPathInfo function in CSPath breaks down this path into a PATH\_INFO struct. This struct provides the Site Id and the path to the file or directory. The GetFullPath and the Combine(PATH\_INFO) methods perform the opposite action by combining the given elements into a Corkscrew Resource Uri.

## Locking, CSLockableObjectBase<T> and the CSLock class

Corkscrew provides the ability for its consumers to lock resources hosted on it. The locking helps the consumer retain the object in an unchanged and known state for a fixed period of time so that it can perform its operations. The entire hierarchy of objects – Farm, Site, Directory and File – can be locked.

Locks are granted either for a fixed duration (3 minutes) or till the lock is released, whichever is earlier. That is, if a lock is not released 3 minutes after it was acquired, it becomes stale and invalid. Other callers can then acquire their own locks or take action on the object.

The CSLockableObjectBase<T> base class allows developers to add locking capabilities to any object in the Corkscrew API. The CSFarm, CSSite and CSFileSystemEntry already inherit from this class. The lock itself is represented by the CSLock class. Locks are taken by calling methods in the CSLock class – the Lock and Release methods. The CSLockableObjectBase class adds the AcquireLock method to help objects take the lock, this method is a wrapper around the CSLock’s Lock method and returns a CSLock object representing the lock that was acquired.

Locks may be taken on a particular object or recursively. A recursive lock effectively locks all its child objects against change. For example, a non-recursive lock on a CSSite only prevents changes to the CSSite object’s metadata. Other callers would still be able to modify its file system. However, a recursive lock on a CSSite instance also locks up its file system and no files or directories in that site can be created, modified or deleted for the duration of the lock. Corkscrew also provides three categories of locks, based on the types of operation that the lock-acquiring consumer wishes to prevent –

* DenyChange – An update operation specific lock that allows the lock-holder to perform modifications but prevents any other user from performing updates. Other users may however delete the object.
* DenyDelete – A delete operation specific lock that allows the lock-holder to perform any operation, and prevents other users from deleting the object. Other users may update the object.

Corkscrew grants locks for a combination of object and user account. A single user account may hold any number of locks at a given point of time. However, an object may have only one type of lock on it at a time. If there is already a DenyChange lock on an object, a request for another DenyChange or a DenyDelete lock will be rejected.

A special case in locking are the locks acquired by workflows. A workflow can run for a long period of time and also pauses and continues based on different triggers. If an object’s creation or modification trigger a workflow instance, the object is locked for the duration that that workflow instance is in a running or runnable state. When the workflow terminates (either successfully or in a failed state), the lock is released automatically.

In supported databases, Corkscrew’s installation scripts also create a database job that runs every 5 minutes and cleans up stale locks. A lock is considered “stale” if –

1. It is older than 3 minutes and;
2. If it is associated with a workflow instance, that instance exists and is in a runnable state.

## Backend connectivity providers, ICSDatabaseProvider and the CSDatabaseProviderFactory

Corkscrew provides an extensible mechanism to allow its clients and systems connect to a variety of backend systems. This means that it is not limited to one or two database engines, but to practically any system capable of surfacing content. Such a backend system could be:

1. A database engine. Microsoft SQL Server and MySQL are already supported out of the box. Developers could extend this to Oracle, PostGreSQL, MS Access or any other database engine.
2. A flat file. Data format files like Xml files, Json files, binary format files and such could be leveraged to store content as well.
3. The operating system’s file system. Since the operations performed in the CMS are similar to operations done on a file system, it is possible to encapsulate a file system (a drive or folder structure on the disk) as Corkscrew’s backend repository.
4. A web service. A WCF or REST API web service could abstract a remote persistence layer. This could enable Corkscrew to leverage cloud storage systems like Microsoft’s OneDrive, Google’s GDrive or other services like DropBox, etc as storage partners.

In order to perform such an extension, the developer should implement the ICSDatabaseProvider interface provided in the Corkscrew SDK. This interface lives in the **aquariusos.corkscrew.library.providers.database** namespace. This interface mandates the following methods to be implemented:

* ExecuteNonQueryStatement\* – to be able to run CRUD operations.
* ExecuteSelectStatement\* – to be able to run simple SELECT operations.
* ExecuteStoredProcedure – used heavily for most operations. See the Appendix at the end of this document for a detailed list of all the stored procedures, their parameters and how to work with them.
* GetAllDatabases\* – to retrieve a list of all the databases on the backend instance.
* GetBinaryContent – used to retrieve the data content of a CSFileSystemEntryFile.
* GetSingleValue\* - used to get a single (column) value from the backend.
* TryConnect – This is used by the CSDatabaseProviderFactory class to check if the provider can connect to the endpoint indicated by a connection string.

\* These functions are not used at all in the Corkscrew SDK code

Regardless of what the actual endpoint is (database, web service, etc), the way to pass that configuration information into the provider you write is through a connection string. The .config file for your application must contain a <connectionStrings> section and within that section, two connection strings. One should have the name “configdb” and the other “sitedb”. The Corkscrew SDK does not check the format of the connection string value, but the provider that needs to connect to it should understand that string. For example, if you are abstracting an OS file system as a backend, that provider’s connection string could simply be “C:\CorkscrewFileSystem” or “/corkscrew/fs” or an smbclient-like option string depending on what the provider understands.

### CSDatabaseProviderFactory

This is a factory class in the **aquariusos.corkscrew.library.providers.database** namespace. It is not an externally callable class. The database layer calls into this class. The way it works is that the first time the backend is accessed, the database layer asks the CSDatabaseProviderFactory for an applicable provider. If the factory has not be called by the AppDomain as yet, it loads up the “configdb” connection string from the config file. It then loads up all the classes in all the loaded assemblies in the AppDomain and looks for classes that implement the ICSDatabaseProvider interface. One by one, each class found is instantiated and then its TryConnect method is called with the particular connection string.

Once a provider confirms acceptance of a connection string, that provider is used for all subsequent connections to both Config DB and all content DBs. To change the provider, one has to restart the AppDomain (as appropriate for the type of running application).

# Features of Corkscrew

Corkscrew provides the following features:

## Content Management

* Workflow support - supports Xaml and Coded workflows written in either WF3 or WF4. Can also use built-in Corkscrew Workflows.
* Out of the box workflow for approval activities.
* Automatic change tracking for Sites, Directories and Files with full preservation of previous data and metadata.

## Search

* Leverages the backend’s search capabilities to provide full text indexing and search.

## Cross Platform

* Targets .NET Framework 4.6.1. Can run on any compatible system on MONO.
  + Cannot run on .NET Core systems due to dependency on workflows (not supported on .NET Core).
* Can work with a variety of backend systems. Out of the box, supports Microsoft SQL Server and MySQL Server.
* Plug-in architecture for integrating backend systems.

## Developer-friendly

* Minimal external dependencies
* Hierarchical object model
* Complete documentation of API, with code examples (this guide)

## Limitations of Corkscrew

Corkscrew has the following limitations on its capabilities and usage.

* Limited to a single database for configuration and content.
  + Advanced users may hack the DB code a bit to get Corkscrew to span to another database for the Site DB. However, in this scenario, Workflows will not function on the sites.
* Cannot run under .NET Core runtime, it requires the full .NET Framework or the MONO framework.

## Collaboration with Corkscrew

In the current version there are no collaboration features directly embedded in Corkscrew except for email. Corkscrew provides for outbound capability. This feature exists primarily for workflows and notification engines in the application. There are no features for communication (like instant messaging or document sharing).

# Code Examples and How to do various things

This chapter provides code samples and illustrations of how to perform various activities in Corkscrew. Note that code samples provided later may refer to or require the code from previous examples.

## Logging into Corkscrew

Access to Corkscrew’s object model and data is secured using its user account and security system. Thus, before accessing an instance of one of its objects, you must login to Corkscrew. To do this, use the CSUser class’ Login method. The method has two overloads.

The Login() instance method can be used if you already have a CSUser instance. The Login(username, password) method is the typical method that you would want to use. To use this method, you start with the username of the account and its corresponding password. The Login method expects that the password argument is passed a SHA256 hash of the actual password. The Corkscrew SDK provides a utility method (GetSha256Hash) to get this value.

On successful login, both Login methods return an instance to the logged in user account or NULL indicating a login failure.

|  |
| --- |
| Code Example |
| string plainTextPassword = “p@ssw0rd”;  string passwordHash = Utility.GetSha256Hash(plainTextPassword);  CSUser loggedInUser = CSUser.Login(“username”, passwordHash);  if (loggedInUser == null) {  // login failure!  } else {  // login succeded!  } |

## Getting a CSUser object

There are multiple ways to get a valid CSUser object.

1. Login to the farm using the CSUser’s Login method as shown in the [above example](#_Logging_into_Corkscrew).
2. The system user accounts (System User and Anonymous) are not backend-persisted entities. For these accounts, you can use the relevant Create methods to create an instance.

To get a system user account instance ---

CSUser systemUserAccount = CSUser.CreateSystemUser();

To get an anonymous user account instance ---

CSUser anonymousUserAccount = CSUser.CreateAnonymousUser();

## Deploying a new farm

A new farm cannot be deployed using the SDK code. The basic steps to create a new farm is to create a new “Corkscrew\_ConfigDB” endpoint as can be accessed through the ICSDatabaseProvider system. If the endpoint is a Microsoft SQL Server or a MySQL database, then the scripts to deploy such a database are available in the Corkscrew distribution. The Corkscrew Explorer UI tool in the distribution helps to automate this task through its UI.

See [this section](#_Setting_up_a) for deployment options.

## Connecting to a farm

Before you can perform any action on Corkscrew, you must connect to the Farm. To do this, use the CSFarm class’ Open method. The method is defined as –

Open(CSUser user)

Or

Open(string username, string passwordHash)

To use the first method, you must already have a CSUser instance. This is true if you have already performed a Login operation or you wish to connect to the farm using a system user credential. The second overload can be used when you have a username and its corresponding password. See the [previous example](#_Logging_into_Corkscrew) for information on how to login to the farm and retrieve a CSUser instance.

As with the CSUser’s Login method, the CSFarm’s Open method also expects a SHA256 hash of the actual password.

|  |
| --- |
| Code Example – Logging in with a CSUser object |
| CSFarm farm = null; string plainTextPassword = “p@ssw0rd”;  string passwordHash = Utility.GetSha256Hash(plainTextPassword);  CSUser loggedInUser = CSUser.Login(“username”, passwordHash);  if (loggedInUser != null) {  farm = CSFarm.Open(loggedInUser);  } |

|  |
| --- |
| Code Example – Logging in with username and password |
| CSFarm farm = null; string plainTextPassword = “p@ssw0rd”;  string passwordHash = Utility.GetSha256Hash(plainTextPassword);  farm = CSFarm.Open(“username”, passwordHash); |

## Create a new Site

New Corkscrew Sites can be created through the CSFarm object only. To do this,

|  |
| --- |
| Code Example |
| CSFarm farm = null; // get CSFarm instance here  CSSite site = farm.CreateSite(name, description, databaseServer, databaseName, quota); |

* name – must be unique in the farm. Site creation will fail if it is not unique.
* Description – optional, this may be shown in user interfaces to let the user identify what the site is, etc. It is not used for any other purpose within Corkscrew.
* databaseServer – the address of the server hosting the site’s database. This should be whatever string the applicable backend provider understands. Note that the database layer will attempt to substitute this value for the “$(DBServer)” token in the connection string if it is present.
* databaseName – the name of the site’s database. Note that the database layer will attempt to substitute this value for the “$(DBName)” token in the connection string if it is present.

**Note: In the current version of Corkscrew, the databaseServer must be the same as the Farm’s ConfigDB server address, and the databaseName must be Corkscrew\_ConfigDB. If a site has a different server and database, then workflows associated to that site or in its file system will be disabled.**

## Connect to a Site

There are two ways to get at a CSSite instance (that is, an existing instance of a CSSite). One is to use the AllSites collection property of the CSFarm object and the other is to use the CSSite’s Open method. Prefer using the AllSites property.

To locate a site within the AllSites property (of type CSSiteCollection), use the CSSiteCollection’s Find method. You may locate a site using its Guid or its name (because names are unique in the farm).

|  |
| --- |
| Code Example – from CSFarm using the site’s name |
| CSFarm farm = null; // get CSFarm instance here  CSSite site = farm.AllSites.Find(name); |

CSSite’s Open method has four overloads and a non-overloaded distinct method. The overloaded methods allow you to fetch the site using either a username and password combination or a CSUser instance, and with either the site’s Guid or its [resource Uri](#_The_Corkscrew_Resource). The fifth method (OpenByDnsName) fetches the site with a CSUser instance and any one DNS name attached to the site – this only works if the DNS name is a non-empty string.

|  |
| --- |
| Code Example – using CSSite’s Open overload |
| CSUser user = null; // get a valid user account instance here CSSite site = CSSite.Open(user, siteId); |

## Set or modify a farm-wide configuration setting

Corkscrew can persist farm-wide configuration into its ConfigDB backend source. To do this, use the CSFarm’s AllConfiguration property. This is a collection of type CSConfigurationCollection, that contains CSKeyPairValue as elements. Each CSKeyPairValue is an object containing a Key (name of the configuration) and Value (its value). The keys can be set up in any nomenclature desired.

|  |
| --- |
| Code Example |
| CSFarm farm = null; // get CSFarm instance here  // adds a new key/value pair with the path “NewConfigurationName” farm.AllConfiguration.Add(“NewConfigurationName”, “ConfigurationValue”);  // modifies the value added above to “NewValue”  farm.AllConfiguration.Update(“NewConfigurationName”, “NewValue”);  // deletes the configuration value  farm.AllConfiguration.Remove(“NewConfigurationName”); |

Settings added or modified are immediately persisted and available to any instance of Corkscrew or its consumer applications connected to that farm. Corkscrew does not reset or stop/start any applications (including workflows) based on configuration changes.

## Create a new user

To create a new user, use the CSFarm’s AllUsers collection and the Add method provided by it.

|  |
| --- |
| Code Example |
| CSFarm farm = null; // get CSFarm instance here  CSUser user = farm.AllUsers.Add(username, displayname, password, email); |

The password parameter of the Add function must be the plain-text version of the password. The Add function will create a SHA-256 hash of this value and store that hash in the backend. The function returns the newly created user.

## Change the password for a user

The CSUser class contains an instance method called ChangePassword. This method has two overloads. The overload with no parameters sets a newly generated random password. The new password is not available (that is, the hash will be stored in the backend, but there is no way to retrieve the original plaintext version). The other overload allows the caller to set a specific plain text password.

|  |
| --- |
| Code Example |
| CSFarm farm = null; // get CSFarm instance here  CSUser user = farm.AllUsers.Find(userId);  if (user != null) {  // set random password  user.ChangePassword();  // set [newP@ssW0rd] as the new password  user.ChangePassword(“newP@ssW0rd”);  } |

## To delete a user

The CSFarm’s AllUsers property (of type CSUserCollection) provides the Remove instance method to allow callers to remove users from Corkscrew. When a user is deleted, all provided permissions for that user are removed too. However, data or workflows owned or modified by that user will not be deleted. There are two overloads for the Remove method, one accepts the user’s Guid and the other accepts the CSUser instance for the user account.

|  |
| --- |
| Code Example |
| CSFarm farm = null; // get CSFarm instance here  farm.AllUsers.Remove(userId); |

## Make changes to the site

To make changes to a CSSite instance, the caller needs to set the required properties and call the Save instance method.

|  |
| --- |
| Code Example |
| CSFarm farm = null; // get CSFarm instance here  CSSite site = farm.AllSites.Find(siteId);  if (site != null) {  site.Description = “New Description”;  site.Save();  } |

## Delete a site

The CSFarm’s AllSites property (of type CSSitesCollection) provides the Remove instance method to allow callers to remove sites from Corkscrew. When a site is deleted, associated data for that site is dropped as well. Any associated workflows are terminated. There are two overloads for the Remove method, one accepts the site’s Guid and the other accepts the CSSite instance for the site.

|  |
| --- |
| Code Example |
| CSFarm farm = null; // get CSFarm instance here  farm.AllSites.Remove(siteId); |

## Create a new directory

A new directory can be created within the file system of a CSSite. The first step therefore is to get access to a CSSite instance and then use the CreateDirectory method on its RootFolder property. RootFolder is of type CSFileSystemEntryDirectory. This class provides two methods – CreateDirectory and CreateDirectoryTree. The differences are as follows:

* CreateDirectory – To use this method, the parent CSFileSystemEntryDirectory instance must be the parent of the new directory as well. That is, this method only creates directories as its immediate child.
* CreateDirectoryTree – This method can be used to create entire sub-trees of directories. It accepts either a rooted or relative path. Directory paths that exist are ignored during creation. If the operation fails in between, all changes made are rolled back.

|  |
| --- |
| Code Example – using CreateDirectory |
| CSFarm farm = null; // get CSFarm instance here  CSSite site = farm.AllSites.Find(siteId); // get CSSite instance  // find a directory named “/foo” in the root folder of the file system.  CSFileSystemEntryDirectory directory = site.RootFolder.Find(“foo”, false);  Directory.CreateDirectory(“bar”); // create a subdirectory named “bar” |

|  |
| --- |
| Code Example – using CreateDirectoryTree |
| CSFarm farm = null; // get CSFarm instance here  CSSite site = farm.AllSites.Find(siteId); // get CSSite instance  Site.RootFolder.CreateDirectoryTree(“/foo/bar”); |

As you can see, the advantage of using the CreateDirectoryTree method is that the calling code does not need to bother with navigating to the correct directory before creating the folder.

## Create a new file

Like a directory, a file can be created within a folder in the file system of a CSSite instance. For this, the CSFileSystemEntryDirectory class provides an instance method called CreateFile. This method has two overloads. One overload creates the file’s metadata and then the calling code needs to write in the data using the file’s stream object. The second overload creates the file’s metadata and writes in the data using the provided byte-array. Both overloads return the newly created file’s instance.

|  |
| --- |
| Code Example – first overload |
| CSFarm farm = null; // get CSFarm instance here  CSSite site = farm.AllSites.Find(siteId); // get CSSite instance  // create a file with the name “filename.extension”  CSFileSystemEntryFile file = Site.RootFolder.CreateFile(  “filename”,  “.extension”  );  // attempt to open the file in Write mode, with no lock  if (file.Open(FileAccess.Write, LockTypeEnum.None)) {  // write the data  file.Write(data, 0, data.Length);  // flush and close the file  file.Close();  } |

|  |
| --- |
| Code Example – second overload |
| CSFarm farm = null; // get CSFarm instance here  CSSite site = farm.AllSites.Find(siteId); // get CSSite instance  // create a file with the name “filename.extension”  CSFileSystemEntryFile file = Site.RootFolder.CreateFile(  “filename”,  “.extension”,  data  ); |

As you can see, the second overload is more compact. However there may be cases where as a developer you need more control with writing the file, or possibly the file’s data is being streamed in from elsewhere (the first overload suits this case better).

## Resolve a Corkscrew Uri and open the correct object

Corkscrew object hierarchy elements do not have a common denominator. This means the SDK cannot provide a single method to retrieve an element from this hierarchy. However, doing this is not difficult – it just requires a lot more code. What needs to be done is as follows:

1. Use the CSPath’s GetPathInfo method to parse the Uri and get its elements.
2. Examine the returned PATH\_INFO’s ResourceUriScope property to find out the deepest element the Uri points to.
3. Use the relevant methods in the SDK to get that element.

Let’s use a Corkscrew Uri of “corkscrew://[00000000-0000-0000-0000-000000000000]/foo/bar”. This points to a file or directory called “bar” within a folder “foo” in the root directory of the farm’s Config site.

|  |
| --- |
| Code Example |
| PATH\_INFO info = CSPath.GetPathInfo(“corkscrew://[00000000-0000-0000-0000-000000000000]/foo/bar”);  switch (info.ResourceUriScope)  {  case ScopeEnum.Farm:  CSFarm farm = CSFarm.Open(user);  break;  case ScopeEnum.Site:  CSSite site = CSSite.Open(user, info.SiteId);  break;  case ScopeEnum.FileOrDirectory:  CSSite site = CSSite.Open(user, info.SiteId);  // get “bar” from the Url  string endPath = Path.GetFileName(info.ResourceUri);  CSFileSystemEntry item = site.RootFolder.Find(  endPath,  true,  (i =>  i.FullPath.Equals(info.ResourceUri)  )  );  break;  } |

In the above code example, you would need to do what you need to do with the resolved object within the switch-case statement just before each break statement.