Developer's Guide

Structural Design

The design of the code is very simple and yet at times convoluted. This is because there have been a handful of developers working on it without any real cohesive vision for the structure of the code. I've done my best to clean things up and create a common flow throughout the code using my personal preferences and conventions but it still has a long ways to go.

Data Sources

Data comes from a few different places. These sources can be broken down into two categories **`System Sources`** and **`Local Sources`**

System Sources

This is where the vast majority of data comes from. The application would not be functional without data loaded from these sources.

- LDAP / AD (Lightweight Directory Access Protocol / Active Directory)
- WMI (Windows Management Instrumentation)

LDAP / AD

The LDAP data source is loaded at launch during the **`LoadForm.Load`** event. It is simply a collection of **`System.DirectoryServices.SearchResult`** objects that has been converted into the domain object that represents a host simply called **`Computer`**. The domain **`Computer`** object is a derivative of the abstract base class **`DataUnit`**. This design allows for polymorphic behavior when loading different types of data into the main data source collection since **`DataUnit`** only requires an implementation of two string properties **`Value`** and **`Display`**. This collection of **`Computer`** objects is loaded into a **`BindingList<DataUnit>`** so that it can be bound to the search combobox on **`MainForm`**.

During the data loading process in **`LoadForm`**, a connection is established directly to any one of the **`Domain Controllers`** is the domain specified by the **`LDAPEntryPath`** application setting (see the Application Settings section for more details). After establishing a connection with a **`Domain Controller`** a query is then executed requesting all **`computer`** LDAP objects with the following attributes (note: by convention, the LDAP protocol uses **`camelCase`** both for classes and attributes)

- name
- description
- uid
- displayName
- extensionAttribute1
- networkAddress
- whenChanged

These attributes may appear to be mostly self explanatory but there are some important things to note about them, especially since a few of them contain data that is custom/proprietary to our Active Directory implementation here at the City of Davenport.

`name`
The hostname which was set automatically when the **`computer`** object was created (when the computer was joined to the domain, or when the **`ComputerName`** is changed).

`description`
Must be set manually after the **`computer`** object was created. This is usually done within ADUC by right clicking > Properties on the computer. SysTool provides a way of setting this field manually as well.

`uid`
Unpopulated by default. We set the **`uid`** value on each **`computer`** object with a **`PowerShell`** script that is ran on each host under the **`SYSTEM`** account. This script is part of a separate project known as **`ADUpdate`**. The value is taken from the **`LastLoggedOnSAMUser`** registry entry on each host located at **`HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI`**.

`displayName`
By default this attribute is populated with the hostname upon creation of the **`computer`** object within Active Directory. Normally, it is never updated again. The **`displayName`** is another attribute that we set automatically with **`ADUpdate`**. The value is taken from the **`DS_displayName`** property of the **`ds_user`** mapped Active Directory class through the local WMI provider.

`extensionAttribute1`
Our Active Directory schema still contains "extensionAttributes" left over from our Exchange 2003 deployment. These attributes exist for the sole purpose of tagging LDAP objects with any custom information. Since these attributes have since been abandoned in current versions of Exchange, we can safely repurpose them to be used in SysTool without having to extend the LDAP schema. **`extensionAttribute1`** holds the LastLogon date/time value and is automatically set by **`ADUpdate`**. The value is simply the current date/time of when the **`PowerShell`** script runs at logon.

`networkAddress`
Unpopulated by default. This attribute normally contains an array value but I have found it is very difficult to control the order that the array[index] is set/get. So instead of trying to rely on the LDAP array getter/setter implementation we set this to a comma separated value in the format of `IPAddress,MACAddress` (for example `10.1.4.197,6C:3B:E5:32:C0:F1`). The values are taken from the `IPAddress` and `MACAddress` properties of the local `Win32_NetworkAdapterConfiguration` WMI class instance. The correct network adapter is found by using the filter `"DNSDomain LIKE '$env:USERDOMAIN%'"`

`whenChanged`
The date/time when the `computer` object was last changed. This is used during the incremental update process. See the LDAP Monitor section for more details.

WMI

The WMI data source is loaded per host upon establishing a connection and also as needed while interacting with the host. WMI is Microsoft's API designed for administering every aspect of a Windows computer system. All of the data displayed within the `Computer Control` comes from WMI except that which was loaded from LDAP at launch.

Local Sources

These sources are for data pertaining to the current user and session.

- Registry
- User Settings

Registry

The Windows Registry is used to store some of the user's custom data including `Custom Collections` (as opposed to `Search Collections`), `Custom Queries` and `Custom Actions`. This data is loaded at launch during the LoadForm.Load event. Currently the only other data being stored in the registry pertains to the User Initiated Upgrade.

User Settings

A collection of hostnames and collections are stored in the user settings called `ActiveComputers` and `ActiveCollections` respectively. See the User Settings section for more details.

Patterns

Software design patterns that I have either noticed throughout the code or have implemented myself.

UI Invocation

Implemented using a set of extension methods on the `System.Windows.Forms.Control` class. This implementation completely eliminates the older repetitive `InvokeRequired` pattern. See this excellent CodeProject article for more details. All extension methods are located in `Modules/Extensions.vb`. See Extension Methods for more information.

Dependency Injection (DIP)

Inversion of Control (IoC) is a modern design concept that often includes some form of DIP. Some progress has been made in implementing DIP by identifying any external dependencies and requiring them to be provided upon object construction/instantiation. This pattern can be seem most notably in the following classes:

- MainForm
- Computer
- DataSourceSearcher
- LDAPContainer

Asynchronous Code

Currently async code is a bit of a mess since there is no consistent pattern used.

Remote Computer Connection Initiation

- Locations: MainForm, ComputerControl
- Pattern: Events (BackgroundWorker)

Mass Import

- Location: MainForm
- Pattern: Events (BackgroundWorker)

Ping Monitor

The Ping Monitor continuously loops through the computer nodes loaded in the **`Resource Explorer`** (note: a **`ComputerControl`** object is tagged to each loaded **`TreeNode`** during the MainForm.LoadComputerNode() procedure). The Ping Monitor code calls the **`ComputerControl.ReportConnectionStatus()`** method on each node. The **`ReportConnectionStatus()`** method then pings the hostname, checks the WMI connection status/speed, and finally sets the **`TreeNode`** text font accordingly (see the User's Guide Connection Status section for more details).

- Location: MainForm
- Pattern: Tasks (Async Methods)

LDAP Monitor

The LDAP Monitor incrementally updates the LDAP Data Source. The **`LDAP Monitor`** knows that a change to the directory has been made by searching through all **`SearchResults`** for the most recent **`whenChanged`** attribute value. It then uses this value to perform another LDAP query to return all **`computer`** objects that have been changed at that time. If a **`computer`** returned in the **`SearchResults`** matches a domain **`Computer`** object already loaded into the LDAP Data Source it is updated. If no match is found then a new domain **`Computer`** object is created and added to the bound data source.

- Location: MainForm
- Pattern: Tasks (Async Methods)

Status Strip Monitor

The Status Strip is a feature that first became available in version 1.6.9. It was implemented mainly to aid in debugging efforts. The Status Strip Monitor runs continuously in order to provide information to the user/developer such as the current date/time, uptime, cpu usage, memory usage and connections count. See the User's Guide Status Strip section for more details.

- Location: MainForm
- Pattern: Tasks (Async Methods)

Update Monitor

The Update Monitor runs continuously and checks the NuGet feed to see if a new version has been advertised. If the Update Monitor sees that a new version has been release it will notify the user by flashing the taskbar icon and setting the title bar text. See the Deployment section and the User's Guide Upgrading section for more details.

- Location: MainForm
- Pattern: Tasks (Async Methods)

WMI Connection Initiation and Querying

- Location: WMIController, QueryControl
- Pattern: Events (BackgroundWorker)

Memory Optimization

- Location: GarbageCollector
- Pattern: Events (BackgroundWorker)

Remote Tool Execution

- Location: RemoteTools
- Patterns: Events (BackgroundWorker), Manual Threading

Remote Task Execution

- Locations: TaskControl, TaskResultControl
- Pattern: Events (BackgroundWorker)

Collection Registry Loader

- Locations: CollectionControl, CollectionOptionsControl
- Pattern: Events (BackgroundWorker)

Tab UI Initialization, Progress Bar and Data Enumeration

- Locations: Tab, ApplicationTab
- Pattern: Events (BackgroundWorker)

Extension Methods

All extension methods are located in **`Modules/Extensions.vb`**

DataTable

CopyToDataTable<T>()

- Explanation: Creates a **DataTable** from a generic collection of objects.
- Source / More Info: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/implement-copytodatatable-where-type-not-a-datarow
- Locations Used: Currently unused but quite useful when needed.

BindingList

AddRange<T>()

- Explanation: Adds the elements of a generic collection to the end of the **BindingList** while only firing the **ListChanged** event once.
- Source / More Info: https://stackoverflow.com/questions/43331145/how-can-i-improve-performance-of-an-addrange-method-on-a-custom-bindinglist
- Locations Used: DataSourceSearcher.New() (makes Dependency Injection Pattern possible)

Form

FlashNotification()

- Explanation: Used to notify the user by flashing the icon on the taskbar.
- Source / More Info: http://www.vbforums.com/showthread.php?367786-Flashing-Taskbar-(Like-MSN-Messanger)-Resolved
- Source / More Info: https://stackoverflow.com/questions/11309827/window-application-flash-like-orange-on-taskbar-when-minimize
- Locations Used: FormMain.SetNewVersionAvailable() (Part of Update Monitor)

Control

- Explanation: User Interface (UI) invocation code to eliminate repetitive **InvokeRequired** checks.
- Source / More Info: https://www.codeproject.com/Articles/37642/Avoiding-InvokeRequired
- Source / More Info: http://www.interact-sw.co.uk/iangblog/2004/04/20/whatlocks

UIThread()
UIThreadInvoke()
InvokeSetText()
InvokeClearControls()
InvokeAddControl()
InvokeRemoveControl()
InvokeCenterControl()

Naming Conventions

Since I began working on this project my naming preferences have changed quite a bit. I have gone through and renamed variables, functions and classes many times as a result. I am fairly happy with the current conventions in place as they are for the most part short, clean and consistent. Here is a list of items to adhere to:

- **Function** names should use the **VerbObject** convention (See here for a good source of proper verbs).
- **camelCase** for all function parameters and local variables.
- **PascalCase** for all **Properties** and **Functions** (note: I prefer **Private Properties** over variables defined in the **Private** / **Module** scope because of Visual Studio's reference counting feature).
- **Private** / **Module** scope variable references should begin with the **Me** or **MyBase** keyword. Do not use an _ or any form of hungarian notation for this.

Security

All actions that can be performed using SysTool already require security clearance that is delegated through Active Directory Users & Groups. With that being said there **is** a small security implementation within SysTool itself.

LDAP Authentication

Upon launch the **LoadForm.Load** event handler first sets up logging and then verifies if the user is authorized to use SysTool. This is accomplished by querying LDAP through the local WMI provider to see if the user belongs to the authorized group. Currently the authorized group name is hardcoded into the application to prevent someone without access to the source code from easily changing it.

Persistence

Application Settings

- LDAPComputerFilter (string)
- LDAPEntryPath (string)
- LogsRemotePath (string)
- LogsLocalPathWithoutSystemDrive (string)
- ITFolderLocalPathWithoutSystemDrive (string)
- RegistryPathCustomActions (string)
- RegistryPathCollections (string)
- RegistryPathQueries (string)

- RegistryPath (string)
- DomainName (string)
- UpgradeScriptPath (string)
- NuGetURI (string)
- NuGetPkgName (string)

LDAPComputerFilter (string)

Enables the ability to define search criteria when returning the computer collection Data Source from LDAP. This string must be in valid LDAP Search Filter Syntax.. The default value for this setting is: **`(&(!name=DUMMY)(name=*))`**

LDAPEntryPath (string)

The path to the domain for pulling Active Directory **`computer`** objects. In ADsPath Format. The default value for this settings is: **`LDAP://DC=citrix-dch,DC=local`**

LogsRemotePath (string)

All logging is done in a centralized fashion to a single log file (this is accomplished using Apache log4net, see the Logging section for more details). This is the setting that determines where this log file will be written to. It can be any valid Windows file path. The default value for this setting is: **`\\fileserver01\itfiles\logs`**

LogsLocalPathWithoutSystemDrive (string)

If the log fails to write to the remote log path then it will be written locally instead. This setting is the local path to write log files to excluding the **`SystemDrive`** since that value is pulled from the environment within the code. The default value for this setting is: **`IT\Logs`**

ITFolderLocalPathWithoutSystemDrive (string)

Whenever IT needs a place to store certain files/tools locally on machines, IT will put them under **`%SystemDrive%:\IT`**. SysTool needs to know that path so that it can look for files such as **`PsExec.exe`**. Again, the **`SystemDrive`** is excluded since that value is pulled from the environment within the code. The default value for this setting is: **`IT`**

RegistryPathCustomActions (string)

Path to the local registry key where **`Custom Actions`** are stored. The default value for this setting is: **`Software\SysTool3\Actions`**

RegistryPathCollections (string)

Path to the local registry key where **`Custom Collections`** are stored. The default value for this setting is: **`Software\SysTool3\Collections`**

RegistryPathQueries (string)

Path to the local registry key where **`Custom Queries`** are stored. The default value for this setting is: **`Software\SysTool3\Queries`**

RegistryPath (string)

Path to the parent registry key where all child keys/values are stored. The default value for this setting is: **`Software\SysTool3`**

DomainName (string)

Needed when querying a computer to find the ethernet adapter that is configured to connect to our Active Directory Domain. The default value for this setting is: **`citrix-dch.local`**

UpgradeScriptPath (string)

Path to the **`PowerShell`** script used to upgrade SysTool to the latest version advertised on the NuGet feed. The default value for this setting is: **`\\fileserver01\itfiles\Software\SysTool\upgrade.ps1`**

NuGetURI (string)

URI to the NuGet feed where SysTool releases are advertised. This is how the Update Monitor knows when a new version is available. The default value for this setting is **`http://tfs/City%20of%20Davenport/_packaging/9e1bb9d1-5fa5-4d46-93fc-b747c423d783/nuget/v3/`**

NuGetPkgName (string)

The name of the NuGet package to check for a new version. This gets appended to the end of the **`NuGetURI`** during the Update Monitor procedure. The default value for this setting is **`it-systemnfos3`**

User Settings

- PsExecPath (string)
- MainFormHeight (int)
- MainFormWidth (int)
- UserSettingsMigrateRequired (bool)

PsExecPath (string)

Path to a local copy of **PsExec.exe**. Before SysTool attempts to launch **PsExec.exe** it first attempts to locate it by looking in a few common places. If found the path is stored in this setting. The default value for this setting is: **C:\ProgramData\chocolatey\lib\psexec\tools\PsExec.exe**

MainFormHeight (int)

When the user adjusts the size of the **MainForm** the new **Height** value is stored here. This happens on the **MainForm.ResizeEnd** event. The default value for this setting is: **630**

MainFormWidth (int)

When the user adjusts the size of the **MainForm** the new **Width** value is stored here. This happens on the **MainForm.ResizeEnd** event. The default value for this setting is: **1030**

UserSettingsMigrateRequired (bool)

The value of this setting is **true** by default (this is the value every time a new version is installed). Then during the **LoadForm.Load** event this setting is checked. If the value is **true** the **User Settings** are then migrated with a call to **My.Settings.Upgrade()**. After a successful migration the value is then set to **false** to prevent another migration upon subsequent program launches.

Logging

Logging is performed by the Apache log4net framework which simplifies writing messages from all SysTool instances to a single centralized log file. The implementation can be found within the **[Global]** module. Configuration for **log4net** is XML based and is stored in the **app.config** file. The path to the log file is set dynamically outside of the standard configuration in **[Global].SetLog4NetFileAppenderPath()** and the path is stored in the LogsRemotePath application setting. Here is a snippet of the XML configuration within **app.config** for reference:

```
<configuration>
    <configSections>
        <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net"/>
    </configSections>
    <log4net>
        <appender name="FileAppender" type="log4net.Appender.FileAppender">
            <appendToFile value="true" />
            <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
            <layout type="log4net.Layout.PatternLayout">
                <conversionPattern value="%date [%thread] %-5level %logger - %message%newline" />
            </layout>
        </appender>
        <root>
            <level value="ALL" />
            <appender-ref ref="FileAppender" />
        </root>
    </log4net>
</configuration>
```

Deployment

The deployment implementation for SysTool consists of a **Team Foundation Server**, **Chocolatey/NuGet** and **Ansible AWX** stack. This is the standard deployment stack used for all in-house software packages here at the City of Davenport.

CI/CD Pipeline

Here is a summary of the current **Continuous Integration** / **Continuous Delivery** pipeline implementation.

- Automated **Build** and **Release** definitions are configured in **Team Foundation Server**.
- A new **CI** package is created and published to the **cod-nuget.ci** feed when the **git** repository is pushed to the **master** branch, a **release** branch or a **feature** branch.
- A new **Release** package is created and published to the **cod-nuget.release** feed when the **git** repository is pushed to a **release** branch.
- The version of the package is based on the branch name (ex. **release/release-1.x.x**) but is also incremented automatically using GitVersion. See the Versioning section for more information.
- The package that is built during the automated **Build** is fully compatible with **Chocolatey**. For this to work all that is required is for the **.nuspec** file to include the **tools** folder which holds the **ChocolateyInstall.ps1**, **ChocolateyUninstall.ps1**, and **init.ps1** scripts (see Chocolatey: Okay How Do I Create Packages for more details)
- The package is now deployable on any machine where **Chocolatey** is installed.
- The installation / upgrade is currently being initiated using one of two different methods. The first method uses Ansible AWX to push the install command to any number of hosts via native **WinRM**. The second method utilizes the User Initiated Upgrade feature that is available starting in version 1.7.6.

Versioning

The SysTool versioning procedure is very simple and straight forward. A new release should be made when any significant change to the code has occurred.

The decision to create a new release is entirely at the discretion of the developer making changes. When the time comes to create a new release build the developer should look at the last **release** branch and note the version within the branch name. The new release version can then be

determined by incrementing the version number by 1. For example if the last **release** branch is named **release/release-1.3.2** then the new release branch should be named **release/release-1.3.3**.

In the event that a new release should be made for a minor bug fix then the version number does not require incrementing. This is possible because the SysTool **Build** definition uses a task called GitVersion to automatically calculate a new version suffix based on the date/time.

NuGet Feeds

There are two NuGet feeds (CI and CD) available for advertising SysTool packages to the network. The CI feed is known as **cod-nuget.ci** and the CD feed is **cod-nuget.release**. Please see the City of Davenport's Team Foundation Server Wiki for more information.

Ansible AWX Method

Ansible AWX is the upstream community driven code source/repository that feeds RedHat Ansible Tower. It is a very simple and easy to use cross platform **Configuration Management** tool that can be invoked through a restful API or command-line interface (which also uses the API). AWX has a very handy web based GUI as well. For more info please see the City of Davenport Ansible AWX Wiki. The City of Davenport AWX Web Management Interface is available at https://awx.ci.davenport.ia.us/

Here is a summary of how SysTool is currently being deployed with AWX:

- AWX automatically creates an **inventory** specifically for SysTool from the **hosts-systool** file stored within the **it-awx-projects git** repository on **Team Foundation Server**
- This **inventory** is configured within AWX with OS specific connection details (i.e. WinRM connection and Kerberos authentication info)
- A **job template** is configured specifically for deploying SysTool that points to a **playbook** which contains all tasks required for installing/upgrading SysTool on the target host(s).
- This **playbook** is also stored within the **it-awx-projects git** repository on **Team Foundation Server**.
- The **job template** is then launched which causes Ansible to connect in parallel to all hosts in the **inventory** and run the **playbook**
- The **playbook** results are then made available through the AWX restful API in JSON format and presented within the web GUI as a **job**.

User Initiated Upgrade Method

This method makes it possible to have a non-invasive upgrade process for the users.

SysTool monitors the **it-systemnfos3** package on the **cod-nuget.release** feed awaiting any available new version. When a new release package is published to the feed a notification is displayed to all users giving them the opportunity to upgrade the application at their convenience.

It is at this time that the registry entry at the path stored in the RegistryPath application setting is set. Upon next launch the **LoadForm.UpgradeApp()** method is called from the **LoadForm.Load** event handler which checks the **UpdateAvailable** registry entry to see if it has been set. If it has been set then a **PowerShell** script at the path stored in the UpgradeScriptPath application setting is launched which then upgrades SysTool using **Chocolatey**. This script will first attempt to install **Chocolatey** if is not already available on the path.