Squirrel UUCP Caller- Adding New Transport Abstractions

OUT OF DATE: everything here was updated in 202209r2 and no longer applies. Things are a lot simpler. Updated documentation to follow

I did make an attempt to create UUCP Squirrel Caller in a somewhat extensible way so that new transport abstractions could be added – if for no other reason I would like to add more myself.

Notice the absence of the term "easily" above – I will be honest and admit things are a bit "all-over-the place", but we'll discuss what's involved below. There's plenty of work to do to clean it up and make it better.

It's assumed that you have downloaded the source from Github and have it open in Visual Studio 2022 on a Windows system.

A Good Portion Of This Is Positively Terrible

I'm not a professional programmer, but I program for fun.

In the Visual Basic 6 days I played around with Visual Studio a lot. Say what you will about Visual Basic but the way it integrates GUI design and code is marvelous. I'm pleased to see Visual Studio 2022 is pretty much the same.

Nonetheless, my familiarity with Visual Basic .NET's object model and OOP features was not strong. During development there was a lot of "learning as I go." Because my knowledge towards the end was greater than the beginning, there is the following: bad code, suboptimal solutions and implementation details, and inconsistencies.

I am likely to work on this from time to time and clean them up, and keep this document up to date as well.

Background Information

1. Global Structure

If you are expecting strict and pure adherence to OOP principles, I apologize in advance for any pain you may experience.

A. Utility Module (Module1.vb)

Functions in `Utility` (`Module1.vb`) are globally available. Below is a brief explanation of some of them as they can be used in code that is added.

Access to Global Settings Object

```
Public CurrentSettings As Object = New Settings
```

This is the instance of a `Settings` object that allows access, loading, and saving of application settings. See `SettingsClasses.vb` for details.

I didn’t like how Visual Basic’s `My.Settings` throws exceptions if somerhing doesn’t exist so I rolled my own.

Debug Output

```
Public DebugMessageMode As Integer = 1
Public DebugMessageEnable As Boolean = True
Sub DebugOut(in_string As String)
```

`DebugOut` outputs messages to the debug console, if DebugMessageEnable is true.

Other modes may be implemented in the future to allow it to write messages to a file or other things.

External Command Execution Functions

```
Function ExecuteCommandInWindow(in_binpath As String, in_args As String) As Integer
Function ExecuteCommand(in_binpath As String, in_args As String) As Collection
Sub CommandConnector(in_workingdir As String, in_binpath As String, in_args As
String, receiver As Object)
```

These functions call external commands.

`CommandConnector` will run a command in a separate thread and call back functions as lines are received. I was hoping it would work to capture Cygwin command output live into a textbox, but it doesn’t work. It is used to capture `uustat`’s output to list active UUCP jobs.

String/Parameter Processing Functions

```
Sub SplitOnFirstSpace(in_string As String, ByRef out_strings() As String)
        'Takes a string that is 2 things separated by a space and puts each
        'thing in its own array element.
        '
        'out_strings() needs to be at least a 2-element string array

Function SplitOnWhitespace(in_string As String) As List(Of String)
        'Converts a string of whitespace-delimited tokens into a list of
        'Strings.

Function SplitBySpaces(in_string As String) As List(Of String)
        'Convert string of space-deliminted tokens to a list

Sub ThrowOnThePile(ByRef in_pile As List(Of String), in_option As String)
        'Converts whitespace delimited items in 'in_option' and appends
        'tokens that don't already exist to the list at 'in_pile'.

Function OptionListPreview(in_option_line As String) As String
        'Takes a list of strings and converts them back into a single string.
        'Intended so we can show the end user contents of a list in dialogs.

Sub ExtractPort(in_string As String, ByRef out_strings() As String, DefaultPort As
Integer)
        'Takes a string that is an address and a port separated by a colon and
        'puts each thing in its own array element.
        '
        'If port is not specified, the supplied default port will be populated
        'in the 2nd array.
        '
        'out_strings() needs to be at least a 2-element string array
```

The above functions can be used to extract data from configuration file lines.

File/Path Functions

```
Function RandStr() As String
        'Return string of 16 pseudo-random characters.
        'Not cryptographically safe.
        'Used to make random port names and key filenames.

Function Slashes(in_string As String)
        'Convert forward slashes to backslashes.
        'Also converts double slashes to single slashes

Function Cyg2WinPath(in_cygpath As String, in_Winpathof_cygwinroot As String) As
String
        'Convert Cygwin UNIX path to Windows path.

Function Win2CygPath(in_winpath As String) As String
        'Takes a Windows path and converts it to a Cygwin UNIX path.
```

The above functions are related to paths and filenames, and mostly bridge unix-style filenames to Windows and vice versa.

Parameter Validation Functions

```
Function Validation_DoesFileExist(in_fullpath As String) As Boolean
Function Validation_DoesDirectoryExist_DontMake(in_fullpath As String) As Boolean
Function Validation_DoesDirectoryExist(in_fullpath As String) As Boolean
Function Validation_callfile_systemname(in_string As String)
        'Check if in_string is a valid system name that would appear in the
        'UUCP 'call' configuration file.
Function Validation_callfile_username(in_string As String)
        'Check if in_string is a valid username that would appear in the UUCP
        ''call' configuration file.
Function Validation_Portfile_portname(in_string As String)
        'Check if in_string is a valid port name
Function Validation_sysfile_systemname(in_string)
        'Check if in_string is a valid system name
        'This should follow the same rules as the names in the call file, so
        'we'll just use the same routine.
Function Validation_settings_option(in_string)
        'Check if Squirrel settings option name is valid
        'Decided it's OK for this to follow the same rules as names in the
        'call file, so we'll just use the same routine.)
```

Lots of functions to validate things. The file and folder validation functions will report to `DebugOut` automatically.

Process

1. A New Class Must Be Defined For Your Transport

Unique parameters of your transport are expected to live in their own class.

This class is required to have:

- a `Public ReadOnly` string variable called `TypeName`.
- optionally, a number of other `Public` variables that represent all the properties of the transport.
- a `New()` constructor that has enough arguments for each property, and is responsible for assigning the arguments to the properties and doing whatever other setup might be needed.

`TypeName` must be a string of alphanumeric characters without spaces, and if an invite file wants to reference it, this is the string that should be used.

Review the bottom two classes, part of Squirrel UUCP Caller, as a reference:

```
Public Class KnownSystemNullTransport
        'Instance of this class should be created if a UUCP system is just in
        'the sys config file as a known system.
        '
        'Properties of a KnownSystemNullTransport
        Public ReadOnly TypeName As String = "KnownSystem"
End Class
Public Class SSHTransport
        'Instance of this class should be created if a UUCP system has an SSH
        'pipe port defined.
        '
        'Properties of an SSHTransport:
        Public ReadOnly TypeName As String = "SSHTransport"
        Public SSHBinPath As String
        Public SSHKeyPath As String
        Public SSHLoginName As String
        Public SSHServer As String
        Public uucicoUsername As String
        Public Sub New(in_SSHBinPath, in_SSHKeyPath, in_SSHLoginName,
in_SSHServer)
            SSHBinPath = in_SSHBinPath
            SSHKeyPath = in_SSHKeyPath
            SSHLoginName = in_SSHLoginName
            SSHServer = in_SSHServer
        End Sub
End Class
```

2. `MakeTransportObject()` Needs To Be Able To Recognize And Create The Transport

A. How `MakeTransportObject()` Works

First, let's discuss what happens before this method is called and the exact expectations of this method.

`MakeTransportObject()` is a method of `Public Class UUCPPort`.

Here's all of it's properties:

```
    Public Class UUCPPort
        'If a UUCP system has a port defined, then one of these objects should
        'be instantiated and associated with the UUCP system object.
        '
        Public Name As String = "[NO_NAME_SPECIFIED]"
        Public port_conflines As New Collection
        Public port_confline_type As String = ""
        Public port_confline_command As String = ""
        Public IsTransportObjectDefined As Boolean = False
        Public TransportObject As New Object
```

`UUCPPort` does have a `Public Sub New(in_name As String)` method but all it does is fill in the `Name = in_name` property.

```
        Public Sub New(in_name As String)
            Name = in_name
        End Sub
```

Other properties besides `Name` are `Public` and expected to be filled in by what instantiates them.

The typical thing going around and instantiating new `UUCPPort`s is `Public Function Config.Refresh()`. It will do this as it's parsing through UUCP configuration text files – and `Config.Refresh()` is called during startup and whenever a new system is defined in the UI.

When does `Config.Refresh()` create `UUCPPort`s precisely? After it has read everything in the UUCP `call` and `Port` file, but not the `sys` file yet.

Then:

- Once it is at that point, `Config.Refresh()` will, for each `UUCPPort` it created, call its `MakeTransportObject()` method.

- `MakeTransportObject()` is then expected to look at its properties, create a new transport object, and point `TransportObject` to it - and also set `IsTransportObjectDefined` to `True`.

`port_conflines` is a collection of lines (as `String`s) from the `Port` text file, and that can be used to determine what transport object to make. Again it should contain any lines found in the `Port` file when `MakeTransportObject()` is called.

B. Extending `MakeTransportObject()`

So because the notion of a transport abstraction is not really part of UUCP, there isn't a pretty way to "find out" what it is from the text file.

You will have to get your hands dirty, dig into this method's code, and add statements that check for things that are only present for the new transport abstraction.

Guidelines and Hints

`port_confline_type` might help you

`port_confline_type` is going to be what's after the `type` option in the UUCP `Port` file.

This can definitely help discern a transport type, but for `type pipe`, which will likely be used for the more interesting transport abstractions, additional inference is needed.

In the unlikely event you can associate a transport abstraction with the port's type, then you can just add a new `Case` to the `Select Case` `port_confline_type` block and perform the steps to extract the parameters (if required) and instantiate the transport object there.

Below and further on we will use the example of defining a "Nifty New" transport, that requires an "endpoint" and a "key" for connection.

This is just for illustration purposes.

```
Public Class NiftyNewTransport
        'Example
        '
        'Properties of a NiftyNewTransport
        Public ReadOnly TypeName As String = "NiftyNew"
        Public Nifty_Endpoint As String
        Public Nifty_Key As String
        Public Sub New(in_Nifty_Endpoint As String, in_Nifty_Key As String)
            Nifty_Endpoint = in_Nifty_Endpoint
            Nifty_Key = in_Nifty_Key
    End Class
    …
    …
    Public Class UUCPPort
        …
        …
        Public Sub MakeTransportObject()
        …
        …
        …

            Select Case port_confline_type
                …
                Case "NiftyNew" 'when "type SOMETHING" is part of the port config
                    DebugOut(">> type is NiftyNew")
                    'Stuff to extract parameters
                    Nifty_Endpoint = {code to extract Nifty_Endpoint from conflines, etc.}
                    Nifty_Key = {code to extract Nifty_Key from conflines, etc.}
                    TransportObject = New NiftyNewTransport(Nifty_Endpoint, Nifty_Key)
                    IsTransportObjectDefined = True
```

Confidence collection and Feeling

When a port's type as listed in the UUCP Port file is pipe, we need to look at additional things to find out which transport abstraction applies, because the pipe type is just saying a command needs to be run to establish the transport for uucico.

So the code that does this works from a Confidence collection.

Each item in the Confidence collection is a pointer to a Feeling object (just a wrapper around an integer), and it's keyed on a unique label that later code uses to switch on.

You will need to add a new label; the label does not have to match the TypeName talked about earlier, but should at least somewhat resemble it.

Here is the beginning of the supplied code that kicks off the process of examining things more closely once becoming aware that the port type is pipe.

```
Case "pipe"
     DebugOut(">> type is pipe")
     DebugOut(">> Attempting to discern transport from command '" &
port_confline_command & "'")
     'Port type "pipe" can be anything.  Need to examine
     'the specified command for clues and discern according to
     'our levels of confidence.
     '
     'Variables for parsing.
     Dim cstart As Integer = 1
     Dim cend As Integer = Len(port_confline_type)
     Dim ctoken As String = ""
     Dim cnextflag As Integer = 4
```

After some initialization, we make a new Feeling for each transport type we might detect from this, and add it to the Confidence list. To extend MakeTransportObject() to recognize the new transport abstraction, a required step is adding a new Feeling dedicated to the transport abstraction to the list of Confidences.

```
     'Tracking our confidences.
     Dim Confidence As New Collection
     Confidence.Add(New Feeling, "ssh")
     Confidence.Add(New Feeling, "niftynew") 'Example
```

(Keep in mind the second argument to Add sets that items key, so we can "lookup" the item by key and not by index later.)

Next in the code is allocating variables to hold parameters we might be collecting.
You can see that happening for SSHTransport.

```
     'We gather data as we see it.  Variables are allocated
     'now for each transport type we attempt to discern here.
     Dim new_SSHBinPath As String = ""
     Dim new_SSHKeyPath As String = ""
     Dim new_SSHLoginName As String = ""
     Dim new_SSHServer As String = ""
     Dim new_SSHPort As String = ""
```

so go ahead and add ones here. I know this is not sustainable longterm. ☺

```
    ‘Example
    Dim new_niftyEndpoint As String = ""
    Dim new_niftyKey As String = ""
```

Because the `pipe` type invokes a command and pipes it to the `uucico` process, the command described in its port description is likely going to have something unique per each transport abstraction that depends on it.

So here in the code we begin looking at the items in the command string and making the `Feeling` of a particular transport’s `Confidence` stronger (or weaker if we wished).

```
    'Let's start going through the command text word by word.
    'cstart and cend are pointers into the string, we extract
    'words based on where we find spaces, then set flags to
    'take actions.
    Do
        cend = InStr(cstart, port_confline_command, " ") - 1
        If cend = -1 Then
            cend = Len(port_confline_command)
            cnextflag = 2
        End If
        ctoken = Mid(port_confline_command, cstart, (cend - cstart) + 1)
        cstart = cend + 2
        DebugOut(">>> token '" & ctoken & "'")
```

At this point is where checks for things that increase a `Confidence` for other transports can be added.

The first `Select Case`/`End Select` block can be used to flag certain parameters as important for the second `Select Case`/`End Select` block.

The second `Select Case`/`End Select` block looks at `cnextflag` and is where `Confidences` are modified. Things that are strongly associated with a particular transport should have stronger `Feelings` applied. If a case applies for multiple transports, a stronger feeling should be applied to each of the `Confidences` associated.

```
        Select Case ctoken
            Case "-i" : cnextflag = 100 : Continue Do
            Case "-l" : cnextflag = 101 : Continue Do
            Case "-p" : cnextflag = 102 : Continue Do
            Case "-endpoint" : cnextflag = 103 : Continue Do
            Case "-key" : cnextflag = 103 : Continue Do
        End Select
        Select Case cnextflag
            Case 1 'beginning of string/first token
                If ctoken = "/usr/local/bin/mkniftynewpipe" Then
Confidence.Item("niftynew").Stronger(10)
                new_SSHBinPath = ctoken
                If new_SSHBinPath = "/bin/ssh" Then
Confidence.Item("ssh").Stronger(10)
                If new_SSHBinPath = "/usr/bin/ssh" Then
Confidence.Item("ssh").Stronger(10)
                If new_SSHBinPath = "ssh" Then
Confidence.Item("ssh").Stronger(10)
            Case 2 'end of string/last token, also time to exit
                new_SSHServer = ctoken
```

```
                 Exit Do
             Case 100 '-i option for SSH key path
                 new_SSHKeyPath = ctoken
                 Confidence.Item("ssh").Stronger(1)
             Case 101 '-l option for SSH login name
                 new_SSHLoginName = ctoken
                 Confidence.Item("ssh").Stronger(1)
             Case 102 '-p option for SSH port
                 new_SSHPort = ctoken
                 Confidence.Item("ssh").Stronger(1)
             Case 103 '-endpoint option for niftynew port
                 new_niftyEndpoint = ctoken
                 Confidence.Item("niftynew").Stronger(10)
             Case 104 '-xxxParam2 option for xxx port
                 new_niftyKey = ctoken
                 Confidence.Item("niftynew").Stronger(10)
         End Select
         cnextflag = 0
     Loop
```

Then, we look and make our decision based on the strongest `Confidence`.

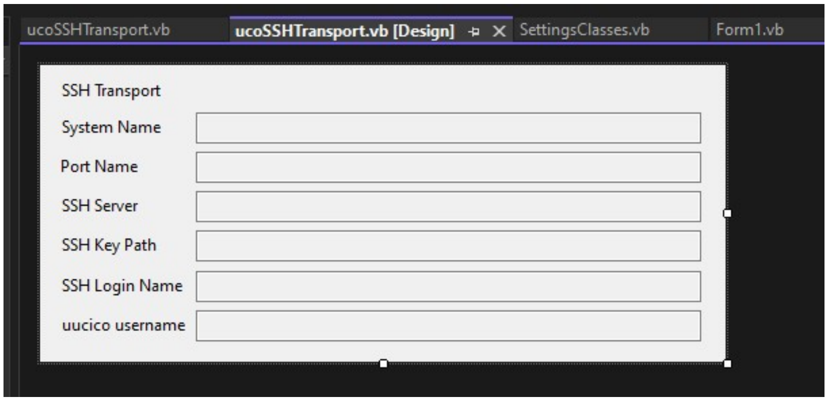
```
     'Find out which type we have the most confidence in.
     'This doesn't handle ties gracefully - FCFS.
     Dim top As Integer = 0
     Dim topKind As String = "meaningless"
     For Each kind As String In {"ssh", "niftynew"}
         DebugOut(">> Confidence that this is '" & kind & "' is " &
Confidence.Item(kind).Level)
         If Confidence.Item(kind).Level > top Then
             top = Confidence.Item(kind).Level
             topKind = kind
         End If
     Next

     DebugOut(">> I have the most confidence that this is '" & topKind & "'")

     'Create object based on what we think.
             'If we don't know and topKind = "meaningless", then we
                    'don't make an object, and don't set
                    'IsTransportObjectDefined.
                    If topKind = "ssh" Then
                        DebugOut(">> Creatng a new SSHTransport for this port")
                        If new_SSHPort <> "" And new_SSHPort <> "22" Then
                            new_SSHServer &= ":" & new_SSHPort
                        End If
                        TransportObject = New SSHTransport(new_SSHBinPath,
new_SSHKeyPath, new_SSHLoginName, new_SSHServer)
                        IsTransportObjectDefined = True
                    End If
                    If topKind = "niftynew" Then
                        DebugOut(">> Creatng a new niftynew transport for this
port")
                        …
```

And of course, hooking into the code in other methods is fine, as long as it doesn't disturb existing scans. This whole system is probably not the best and work may be undertaken to make it better.

3. Create a “Display” User Control Object For The Transport Abstraction

Keep in mind the form will need to fit beside the list of systems in the Systems tab.

Below is a screenshot ucoSSHTransport, part of the code already, as an example.



A. *SetFields(…)*

The User Control Object needs to define a `SetFields` sub that takes all the parameters it displays. This function is responsible for populating the User Control fields.

See how the `ucoSSHTransport` User Control object class does it:

```
Public Class ucoSSHTransport
    Public Sub SetFields(in_systemName As String,
                         in_portName As String,
                         in_SSHServer As String,
                         in_SSHKeyPath As String,
                         in_SSHLoginName As String,
                         in_uucicoUserName As String)
        …
        tbxSystemName.Text = in_systemName
        tbxPortName.Text = in_portName
        tbxSSHServer.Text = in_SSHServer
        tbxSSHKeyPath.Text = in_SSHKeyPath
        tbxSSHLoginName.Text = in_SSHLoginName
        tbxuucicoUsername.Text = in_uucicoUserName
    End Sub
```

4. Create a “Builder” User Control Object – Responsible For Defining UI, Validating, And “Etching” A New System That Wants To Use The Transport Abstraction’s UUCP Port

Expected Properties, Methods And The **New()** Sub

The User Control object that provides the define/validate/persist capability is expected to have these public variables:

```
Public Modified As Boolean = False
Public valid As Boolean = False
Public Caller As Object = Nothing
```

- `Public Modified As Boolean` communicates to external code if any existing data was changed.

- `Public valid As Boolean` communicates to external code whether or not the data in the control’s fields represents a valid instance.

- `Public Caller As Object` is meant to point to the form calling.

    - Also: Whoever the `Caller` is is expected to implement `PanelChange(not_used As String)`.

    - The purpose of this is to allow the User Control to report to its caller that something has changed. This is used by `frmNewSystem` to disable the list of transport types after the user has begun editing one (the user must cancel or move forward and save).

- The User Control object should also include any additional properties/variables needed, and the `New` sub needs to accept parameters that will provide it all the information it needs to define an instance’s data and possibly populate those variables.

- Code that instantiates this object will need to know what this object’s `New` method wants, up and above a pointer to itself.

Below is an example in code:

```
Public Class ucoNewSystem_NiftyNewTransport
    Public Caller As Object = Nothing
    Public Winpathof_etcuucp As String = ""
    Public NiftyEndpoint As String = ""
    Public NiftyKey As String = ""
    …
    Public Sub New(in_caller As Object, in_NiftyEndpoint As String, in_NiftyKey As
Object)
        ' This call is required by the designer.
        InitializeComponent()
        ' Add any initialization after the InitializeComponent() call.
        Caller = in_caller
        NiftyEndpoint = in_NiftyEndpoint
        NiftyKey = in_NiftyKey
    End Sub
```

A. “System Hashtables”

A hashtable is used to pass around data that a builder User Control can use when creating a new UUCP system and it’s UUCP port. Adding a new transport requires some functions to be provided that create and work on those hashtables.

- I’m calling this a “system hashtable.”
- I’m also calling the process of committing a new system to a file “etching.”

The keys in the hashtable are basically strings that match the options as they would appear in the invite file. Look at an example invite file and review the UUCP Invite File And Transport Abstraction Specification to understand further. Note: You’ll be extending that specification as well.

Keys that must appear in all system hashtables are:

`transport-type` internally recognized name of the transport.

`system-name` name of new system, as it will appear in the list of systems in the System tab

> Existing code uses two additional keys, but they don’t correspond to things that should be in the invite file. These keys are intended to indicate processing status:
>
> `failed` this should be a Boolean - `True` or `False`. The purpose of this will make sense when you look at the `MakeFromUI()` expected public function below.
>
> `whyfailed` if `failed` is `True` then this string should contain a textual explanation.

B. Expected Methods

External code (which will be `Public Class frmNewSystem`) will use these functions when building a new system, and your builder User Control must implement them.

MakeFromUI()

This function…

```
Public Function MakeFromUI() As Hashtable
```

should return a system hashtable built from UI elements. This function is responsible for setting the `transport-type` key properly.

CheckIfValid(…)

This function…

```
Public Function CheckIfValid(in_NewSystemHashtable As Hashtable) As String
```

will take an existing system hashtable and verify that all keys are valid.

`CheckIfValid` should return a null string if there isn’t any problems, otherwise the string should contain a textual explanation (optionally containing new-line separated text) describing the problems that external code can use in a dialog.

EtchInFiles(…)

This function…

```
Public Function EtchInFiles(in_NewSystemHashtable As Hashtable, in_conf As Object)
As Boolean
```

will take an assumed-valid system hashtable and do what is necessary to set it up in the UUCP configuration text files. It will need a pointer to a valid `Config` object to get configuration file paths and utilize methods to assist in modifying files.

`EtchInFiles()` should return `True` if successful and `False` if not.

TODO Note: Separate logic that writes the port (which is what the transport is concerned with) from the logic that writes the new system and new username/password to the call file.

C. Implementing Your ***EtchInFiles(…)*** Function

Backup And Prepare For Recovery

Writing A New Port In The `Port` File

This ought to be fairly easy because a new `port` option and its related options can simply be appended to the file.

Writing A New System In The `sys` File

So above, we didn't talk about forwarding options – currently I think it's better that the end user does not have to worry about forwarding while entering all the details to create a new system. So I think it's a good idea for the UI elements in the User Control object to not worry about them.

However, **your builder User Control will be called if the user opens an invite file, and the invite file's `transport-type` matches the TypeName of your new transport abstraction.**

So **`EtchInFiles(…)`** needs to be prepared to deal with it.

Also currently: forwarding options are set globally. That means they need to appear at the top of the `sys` file.

So the **`Config`** object has some functions to make "inserting" things into the `sys` file easier.

- An assumption is that the `sys` file will never be so large that reading it all in RAM isn't a good idea. I feel this is a pretty safe assumption, but some limit should be in place to prevent malicious files/edits from causing problems.

```
Public Class Config
    …
    Public Function PrepareToModifySys(ByRef in_bufferRef As String) As Hashtable
```

`PrepareToModifySys(…)` is explained pretty well in the following comments:

```
'So basically what this function does is:
        '
        '- Read the entire sys file into in_bufferRef
        '   The related Commit function will want it and will use to to
        '   rewrite the file on commit.
        '
        '- Extract certain options into a hashtable - right now, options
        '  relating to forwarding that appear before the first 'system'
        '  block.
        '  *One other thing is put into the hashtable - a value called
        '   "boundary" that tells what line number the system blocks
        '   start in the file.  Hashtable options change "defaults" which
        '   apply to all system blocks and have to appear before any of
        '   them.
        '  *External code can add/remove/change the hashtable and the
        '   commit function will add those lines, or update them in place
        '   if found.
```

So this function will read in the whole `sys` file into a string variable you've put aside – which the calling function is expected to keep around and keep up with – and will additionally give you hashtable with some options extracted into it.

The idea is that you can modify the hashtable, and modify/append to your buffer string if you want, and then the `CommitModifiedSys(…)` function will write back the changes.

```
Public Class Config
    …
    Function CommitModifiedSys(ByRef in_BufferRef As String, ByRef
in_SysGlobalOptions As Hashtable) As Boolean
```

To handle forwarding options from the invite file, the `ModifyForward(…)` sub is provided and it will modify the forward options in a hashtable generated by `PrepareToModifySys(…)`.

```
Public Class Config
    …
    Public Sub ModifyForward(ByRef in_SysGlobalOptions As Hashtable,
                             in_Option As String,
                             in_enableOrDisable As Boolean,
                             in_systemName As String)
```

D. Looking at ***`ucoNewSystem_SSHTransport.EtchInFiles()`*** In Detail As An Example

We'll walk through the SSHTransport's `EtchInFiles()` code in detail – to show exactly what the code does and where.

Getting Internal Variables Ready

First order of business is getting all our internal variables setup – pulling from the hashtable and also randomly generating our Port name.

```
        'Port name
        Dim portname As String = "SSH-" & RandStr()
        Dim NewSSHKeyPath = "/etc/uucp/" &
My.Computer.FileSystem.GetName(in_NewSystemHashtable("private-key-file"))
        Dim NewSSHUserName = in_NewSystemHashtable("ssh-username")
        Dim NewSSHSystemName = in_NewSystemHashtable("system-name")
        Dim NewuucicoCallUserName = in_NewSystemHashtable("uucico-call-username")
        Dim NewuucicoCallPassword = in_NewSystemHashtable("uucico-call-password")

        Dim PortServerSplit(2) As String
        ExtractPort(in_NewSystemHashtable("ssh-server"), PortServerSplit, 22)
        Dim NewSSHServer = PortServerSplit(1)
        Dim NewSSHPort = PortServerSplit(2)
```

Backing Up Existing Files And Enabling Recovery Setting

You should do that. I haven't done that yet. This is a big TODO.

Check for An Existing System

We really need to find out if system already exists by that name in the UUCP configuration text files, and deciding what to do if it does (overwrite or cancel).

This really would be better handled outside of this function - future versions will handle this better. As it is at the moment though, the first thing you should check is if the new system name matches an existing system, and error out. The comments below are inaccurate.

```
Public Function EtchInFiles(in_NewSystemHashtable As Hashtable, in_conf As Object)
As Boolean
    …
    'At this point we need to examine the current configuration and
    'determine if we are going to...
    'A. Append port/system to existing Port/sys conf files
    'B. Overwrite existing system in sys file, add new port to Port file
    'C. Do nothing because it already exists.

    'Check existing systems in config
    Dim ExistingSystemFound As Boolean = False
    For Each ExistingSystem In Systems
        If (ExistingSystem.Name = NewSSHSystemName) Then
            ExistingSystemFound = True
            Exit For
        End If
    Next
    If ExistingSystemFound Then
        SquirrelComms.Item(16).UserError()
        Exit Function
    End If
```

Now comes the fun of actually adding the system to the UUCP `Port` file. Because we have a randomly generated port name and don't need to worry about duplicates (TODO: we probably should check anyway), adding ports to the file is simple and can be done with `My.Computer.FileSystem.WriteAllText(…)`.

```
    'Add new SSH transport based system to our config file.
    '
    'Append new port definition to Port file.
    Try
        My.Computer.FileSystem.WriteAllText(
            Slashes(Winpathof_etcuucp & "\Port"),
                vbLf &
                "port " & portname &
                vbLf &
                "type pipe" &
                vbLf &
                "command /bin/ssh -a -x -q" &
                    " -p " &
                    …
                    vbLf,
                    vbTrue) 'true here means append, false or absent means
overwrite
            DebugOut(" Success: successfully appended new port " &
                     Chr(34) & portname & Chr(34) &
                     " to " &
                     Chr(34) & Winpathof_etcuucp & "\Port" & Chr(34)
                     )
        Catch
            SquirrelComms.Item(19).SystemError("")
            DebugOut(" ERROR: FAILED while appending new port " &
                     Chr(34) & portname & Chr(34) &
                     " to " &
                     Chr(34) & Winpathof_etcuucp & "\Port" & Chr(34)
                     )
            EtchInFiles = False
            Exit Function
        End Try
```

And here is where we use the `PrepareToModifySys(…)` function, handle any forwarding options present in the hashtable with `ModifyForward()` off of the pointer to `Config` that the builder User Control was given in the `New(…)` function. The text that defines the new system is appended to the `SysFileBuffer` string.

```
        'Append to new system to sys file
        Try
            Dim SysDefaultOptions As New Hashtable
            Dim SysFileBuffer As String = ""
            SysDefaultOptions = in_conf.PrepareToModifySys(SysFileBuffer)

            If in_NewSystemHashtable.Contains("please-forward-from-me") Then
                in_conf.ModifyForward(SysDefaultOptions, "forward-from", True,
NewSSHSystemName)
            End If
            If in_NewSystemHashtable.Contains("forward-to-me") Then
                in_conf.ModifyForward(SysDefaultOptions, "forward-to", True,
NewSSHSystemName)
            End If

            SysFileBuffer &=
                vbLf &
                    "system " & NewSSHSystemName &
                    vbLf &
                    "call-login *" &
                    vbLf &
                    "call-password *" &
                    vbLf &
                    "time any" &
                    vbLf &
                    "chat " & Chr(34) & Chr(34) & " \d\d\r\c ogin: \d\L word: \P"
&

                    vbLf &
                    "chat-timeout 30" &
                    vbLf &
                    "protocol i" &
                    vbLf &
                    "port " & portname &
                    vbLf
            in_conf.CommitModifiedSys(SysFileBuffer, SysDefaultOptions)

            DebugOut(" Success: successfully appended new system " &
                Chr(34) & NewSSHSystemName & Chr(34) &
                " to " &
                Chr(34) & Winpathof_etcuucp & "\sys" & Chr(34)
                )
        Catch
            SquirrelComms.Item(19).SystemError("")
            DebugOut(" ERROR: FAILED while appending new system " &
                     Chr(34) & NewSSHSystemName & Chr(34) &
                     " to " &
                     Chr(34) & Winpathof_etcuucp & "\sys" & Chr(34)
                     )
            EtchInFiles = False
            Exit Function
        End Try
```

Then we do need to get the uucico username and password in the `call` file. That's a simple append just like the `Port` file.

```
        'Append username/password to call file
        Try
            My.Computer.FileSystem.WriteAllText(
                    Slashes(Winpathof_etcuucp & "\call"),
                    NewSSHSystemName &
                    vbTab &
                    NewuucicoCallUserName &
                    vbTab &
                    NewuucicoCallPassword &
                    vbLf,
                vbTrue) 'true here means append, false or absent means overwrite
        Catch
            SquirrelComms.Item(20).SystemError("")
            DebugOut(" ERROR: FAILED while appending new username " &
                     Chr(34) & NewuucicoCallUserName & Chr(34) &
                     " to " &
                     Chr(34) & Winpathof_etcuucp & "\call" & Chr(34)
                     )
            EtchInFiles = False
            Exit Function
        End Try

        EtchInFiles = True
```

Clear Recovery Setting.

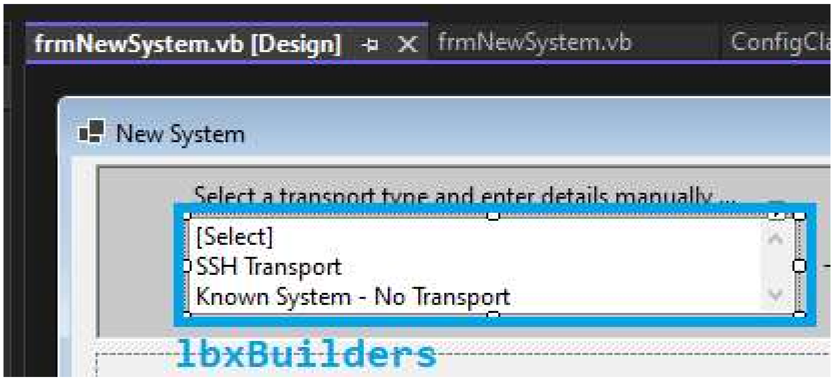
Another thing I talked about but didn't to. TODO. I would leave the old backup files in case the user wants to revert.

5. Extending **frmNewSystem** to Recognize The New Transport Abstraction In The UI.

Prepare for more ugliness.

A. Add A New Item To **lbxBuilders**.

This is **lbxBuilder**s, a list box containing one line for each possible builder. For the new transport abstraction, a line will need to be added here – and you will need to add new ones on the bottom and not disturb existing ones.



Select a transport type and enter details manually ...

[Select]
SSH Transport
Known System - No Transport

B. Add A New Case To lbxBuilders_SelectedIndexChanged(…) To Bring In The Transport Abstraction's Builder User Control.

Basically, just tack on a new case here, where the integer of the Case statement matches the index of the item in lbxBuilders.

What you do in the case is set the Builder pointer to a new instance of your User Control, passing along any parameters from frmNewSystem that the User Control needs.

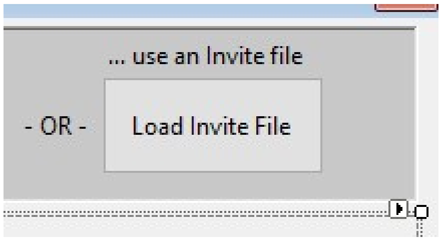
Here’s an example:

```
            Select Case lbxBuilders.SelectedIndex
                Case 1
                    'ucoNewSystem_SSHTransport needs access to a few items in
                    'our config object to perform validation.  These are
                    'passed via New.
                    '
                    'Honestly probably should just pass the whole object.
                    Builder = New ucoNewSystem_SSHTransport(
                        Me,
                        conf.Winpathof_etcuucp,
                        conf.Ports,
                        conf.Systems)
                Case 2
                    'ucoNewSystem_KnownSystem
                    '
                    Builder = New ucoNewSystem_KnownSystem(
                        Me,
                        conf.Winpathof_etcuucp,
                        conf.Ports,
                        conf.Systems)
                Case 3
                    'ucoNewSystem_NiftyNewTransport
                    '
                    Builder = New ucoNewSystem_NiftyNewTransport(
                        Me,
                        conf.Winpathof_etcuucp,
                        conf.Ports,
                        conf.Systems)
            End Select
```

Later code will add what's pointed to by the `Builder` to the panel, making the control appear and be useable in the UI as soon as the user selects it in `lbxBuilder`.

C. Also A New Case To `btnInviteFile_Click(…)` To Bring In The Transport Abstraction's Builder User Control.

This code is called when the user presses the Load Invite File on the UI.



After the user selects a file, and `MakeFromInvite(…)` is called and successfully creates a system hashtable from the invite file, the code needs to create a builder User Control so it can call its functions.

A `Select Case` structure is used here to determine which User Control to create, switched on the text of the `transport-type` option from the hashtable (which is from the invite file).

See example below. The code that would be added would be the exact same code added to `lbxBuilders_SelectedIndexChanged(…)` above.

Here's an example:

```
            Select Case LCase(NewSystem("transport-type"))
                Case "knownsystem"
                    Builder = New ucoNewSystem_KnownSystem(
                        Me,
                        conf.Winpathof_etcuucp,
                        conf.Ports,
                        conf.Systems)
                Case "sshtransport"
                    'ucoNewSystem_SSHTransport needs access to a few items in
                    'our config object to perform validation, as well as this
                    'object to make reactive UI changes.  Passed via New.
                    Builder = New ucoNewSystem_SSHTransport(
                        Me,
                        conf.Winpathof_etcuucp,
                        conf.Ports,
                        conf.Systems)
                Case “niftynewtransport”
                    'ucoNewSystem_NiftyNewTransport
                    '
                    Builder = New ucoNewSystem_NiftyNewTransport(
                        Me,
                        conf.Winpathof_etcuucp,
                        conf.Ports,
                        conf.Systems)
                Case Else
                    SquirrelComms.Item(23).SystemError("The transport type in the
invite file was " & Chr(34) & NewSystem("transport-type") & Chr(34))
                    Exit Sub
            End Select
```

6. Add Your Transport Abstraction To The Invite File Specification.

What you add should follow the format of the specification and document all names and parameters your transport abstraction uses.

The Invite File Specification is located here:

https://github.com/lawrencecandilas/craziness/tree/main/UUCP%20Invite%20File%20Specification