Ratchet-X Desktop RPA Platform

Documentation – Series 6
Contents

The Ratchet-X Platform: 50,000 foot View ................................................................................................... 3
Requirements and Compatibilities ............................................................................................................... 9
Production Changelog .................................................................................................................................. 10
Commander Options Reference .................................................................................................................... 15
Sample Integrations .................................................................................................................................... 23
Licensing: Your license file and how it is used ............................................................................................ 28
Templates: What are they and how do I use them? Can I create my own? .............................................. 36
XModels: How can I make my action independent of my target screen? .................................................. 42
Magic Button configuration: Titlebar, top, right and custom placement....................................................... 51
Conditional Markers vs. Anchor Markers ................................................................................................... 54
Connectors: Custom coding inside your appspace ..................................................................................... 57
Scrape and OCR: When accessibility fails you............................................................................................. 59
Scanner Log: Which Regwins are found and why some are not found? .................................................... 66
Magic Button: Why is it not showing? ........................................................................................................ 68
Java: Yep, we have that covered too .......................................................................................................... 74
Start By Scenarios: Is there an alternative to the Magic Button? ............................................................... 78
Packaging and redistributing a Ratchet-X solution ..................................................................................... 92
Commander Status Bar: What do those characters at the bottom of Commander mean? ..................... 104
Snaggles: Skip the Regwin by finding patterns on any screen ................................................................ 106
Sounds: Please tell me how to turn them off ........................................................................................... 109
My Configuration File: What is it? Where is it? How do I save and share it? ........................................... 110
ClickIts: When an action needs to visit many screens to do its thing ....................................................... 111
Finding Bitmaps: When all you have is an image ...................................................................................... 119
Action Configuration: How to get end user data like passwords and server names ................................ 124
Aggregation Workstation: What is it? Why may I need it? ...................................................................... 126
64bit support demystified ........................................................................................................................ 127
RegWin Debugger: What is it? How do I use it? ....................................................................................... 129
What is MSAA and why should I care? ...................................................................................................... 133
Screen Simulator: Seeing is believing ....................................................................................................... 136
What is a “fingerprint” and how do I tie back to a Ratchet-X license? ...................................................... 149
RatchetSoft Knowledge Base

What Is The Sample Code Editor? ............................................................................................................. 151
MyScreenTester ......................................................................................................................................... 154
Project Methodology and Best Practices ............................................................................................... 156
The Ratchet-X Platform: 50,000 foot View

Ratchet-X is a desktop application integration platform that allows you to add new features to existing applications without requiring changes to those applications or cooperation from vendors. These new features can range in form from a simple push of data from one application screen to another, through highly choreographed, multi-data source syncs, data extractions and pastes. Via Ratchet-X you can integrate the data contained within your application screens with the following data sources:

- other screens in the same application
- other screens in different applications
- web sites
- web services (REST, SOAP)
- databases and structured files (XML, CSV, fixed-length record, etc)
- electronic forms
- custom APIs

Ratchet-X’s ability to recognize any application screen, in any state, coupled with its ability to extract and paste data to and from any data source, makes Ratchet-X a tremendously powerful desktop integration tool.

Ratchet-X Architecture

Before you dive in and start building integrations, we recommend you invest a little time to understand the Ratchet-X architecture. The following architectural primer introduces you to the most important parts of the platform and how they interact. Once you understand these concepts, feel free to jump right in and start building.
The most important architectural components of the Ratchet-X platform are:

- Commander
- Appspaces
- Actions
- Xmodels

**Commander**

Commander is the runtime software for the platform. This is the software that gets installed on each user's desktop. Commander allows you to:

- register appspaces
- register actions
- set Commander configuration options and preferences
• load other platform tools and editors such as: Appspace Editor, Appspace Wizard, Action Editor, Xmodel Editor and all sample applications such MyCRM and MGA Express.

Depending upon how Ratchet-X is configured by your system administrator, it may run in either Standard mode or Compact mode. Standard mode allows you access to all Commander system features and tools. Compact mode merely allows you to execute pre-configured integrations (called Actions). You can configure the mode within which Commander runs by setting the Commander Option CompactModeOnStartup. For more on Commander's Options, click here.

Appspaces
Appspaces are XML files created using either the Appspace Editor or Appspace Wizard. Appspaces are responsible for the following functions:

• informing Commander on how to recognize a specific screen or section of a screen (screens and screen sections are referred to as regwins, or registered windows, within the platform)
• defining how data is extracted from, or pasted to, "fields" contained within regwins (these fields are referred to as snippets)
• defining how UI elements can be manipulated to either change regwin state, navigate to other regwins, or execute an in-application process
• map data from snippets to xmodels
• define keyboard or mouse-based shortcuts that allow a user to execute a specific action from a regwin

An appspace can contain more than one regwin. Most often, all the regwins related to a given user application or all the regwins involved in a multi-application transaction are placed within one appspace. However, there is no hard and fast rule regarding how you should organize your appspaces.

Once an appspace is created, it must be registered with Commander in order for Commander to start scanning the desktop for any of the regwins contained within.

Actions
Actions represent the functions Ratchet-X can perform for you when Commander recognizes a regwin. Actions, and the functions they make available within your applications, “are” the reasons you use Ratchet-X in the first place. Actions are developed using the Action Editor and are written in
either VB.NET or C#. These pieces of code are stored as XML files and compiled on demand by Commander when needed. Further, you can also embed DLLs written in any programming language of choice into an action. Actions are most often presented to the user via the Ratchet-X Magic Button but can be executed a number of ways such as: via mouse or keyboard shortcuts, timers, file drag and drop, etc.

Once an action is created, it must be registered with Commander in order for Commander to make the action available at runtime when specific regwins are found.

**XModels**

Most actions involve the shuttling of data between application screens and actions. The xmodel is the vehicle by which this data is defined and transferred. Xmodels, creating using the Xmodel Editor, is a data structure used by Ratchet-X to house data related to the execution of an action. While an xmodel is usually created to support a specific action, xmodels can also be created in the abstract to define a commonly used piece of data that can be generically used by future actions and appspaces. This decoupling of xmodels and specific appspaces and actions is what allows Ratchet-X to scale. For example, you can create an xmodel that describes a US mailing address (address line 1, address line 2, City, State, Zip Code) in the abstract so it can be used by applications that contain addresses (CRM, accounting, etc,) and actions that can process addresses (address verification, mapping and geocoding, shipping, etc.).

**Putting It All Together**

So how does this all come together in a real world, Ratchet-X project? Let's say we want to build an integration between a CRM system screen that contains a US mailing address and a web service that verifies a US mailing address. The following describes, at a high level, the process you would typically go through to create the integration and how it works at runtime.

1) Identify the application screen that contains a mailing address.
2) Determine which fields on the screen house the original address and which fields will receive the verified results (in examples like this, the collection of fields is usually the same).
3) Determine the inputs and outputs associated with the web service that will verify the address.
4) Create an xmodel using the Xmodel Editor that describes the required data for the address verification transaction.
5) Create an appspace using the Appspace Editor (or Appspace Wiard), that defines the screen which contains the address fields. This screen is defined in Ratchet-X as a regwin (or registered window). Creating the appspace usually involves the following:

   a. How will Ratchet-X recognize the application screen, in the desired state, that contains the address data fields? This process involves defining application window parameters (e.g. process name, title bar text and markers).
   b. Which fields in the screen house the address data? These fields are referred to in Ratchet-X as snippets.
   c. Create connectors that know how to extract data from, and paste data to, the defined snippets.
   d. Map the snippets to the appropriate xmodel elements.
   e. Define any mouse or keyboard shortcuts you might want to use to execute the address verification action (as an alternative to the magic button).
   e. Save and register the appspace in Commander's Appspace tab.

6) Create your action in the Action Editor. This involves writing the C# or VB.NET code that does whatever you need the action to do. In our example, the action call the address verification REST-based web service using the data passed into the action via the mapped xmodel as inputs and writing the verified address back to the xmodel upon completion).

7) Save and register the Action in Commander's Action tab.

Now that the appspace and action are registered with Commander, what happens at runtime? Commander starts scanning the desktop, looking for the appearance of the CRM application screen described in the registered appspace. When the user navigates to that screen, Commander detects that it's the active window and places a magic button in the CRM application's title bar. When the user clicks the magic button, the address verification action is executed using the address extracted from the screen as inputs. When the web service contained within the action returns the verified address, the data is passed back to the appspace and pasted into the address fields.

If you would like to see this use case in action, you can do so by loading the sample assets and MyCRM application that ship Ratchet-X. For more details on this sample, click here.

While this simple use case barely scratches the surface with regard to all the things Ratchet-X can do, it does provide a concise description of a basic Ratchet-X usage pattern. If you understand this, you're ready to dive in and start building your own integrations using the remainder of this
knowledge base as a helpful reference that covers both the "how’s" and just as importantly, the "why’s" of Ratchet-X.

If you come across an issue not covered in this knowledge base or have a suggestion on how to improve the product, please contact us at any of the contact point listed above.
Requirements and Compatibilities

The following details Ratchet-X Commander’s desktop requirements and software compatibilities.

**Ratchet-X Commander Desktop Requirements:**

- Local computer disk space: 75MB (this includes all sample applications, assets and extensions).
- 1 GB RAM.
- Microsoft .NET Framework versions 3.5 and or higher.

**Ratchet-X Compatibilities:**

- Windows XP operating system or higher.
- Browsers: Internet Explorer versions 6 and higher.
- Google Chrome (preferably running in accessibility mode).
Production Changelog

Production Release of Ratchet-X Version 5.5.0.403

Version: 5.5.0.403
Date: 08/14/2015

Notes: This version of Ratchet-X contains the following changes:

1. New ScreenLink methods to click mouse using Windows API introduced. These methods are SendMouseClick and PostMouseClick.
2. Moved some application manifests from internal (inside the EXE), to external files (exe file name plus .manifest). This was done to allow future privilege elevation and uiAccess=true. EXEs include AppspaceEditor, ActionEditor and Commander.
3. Fixed bug in Package Editor which caused a crash when icon file was not found.
4. Added two new fields to MyCRM Win/Legacy/Web... PeopleTab.SSN and CompanyTab.DUNS#.
   Updated the appspaces to account for the change.
5. Appspace Wizard changed to provide default name of XModelID + 4 digit random number instead of simply App+Screen. This change allows the user to change the Application and Screen values without forcing a change to the appspace key. It also allows the user to create many wizard appspaces for the same application screen. Prior to this fix, any attempt to create an appspace using the same values for Application and Screen (as one already registered) would result in a user message saying it was already registered.
6. Changed MyCRM Windows to include First and Last Name in the titlebar text. This will be helpful to demonstrate why regex is needed for titlebars.
7. To assist remote debugging, a new Option is now available in the Commander options dialog box called DebugOptions. This string contains either the value "None" or a collection of tokens (like "Abc" separated with commas). These tokens are not published and this feature is for use strictly by support personnel when attempting to isolate problems out in the field.
8. A magic button tracking issue has been resolved. This bug affected any 64 bit window and IE10 or IE11 only when the user had unchecked the Windows performance option "Show contents of window while dragging". In this scenario, Ratchet-X may not repaint the button after a drag until after the user clicks on the target window frame, leaving it where it was... just hanging in space over the desktop.
9. Custom ClickIts are now possible by overloading the CustomClick method of the ConnectorRuntime object in Appspace Editor.
10. Fixed a bug with Appspace Wizard Tooltip truncating some text on high DPI machines. Change included a slight visual change to the tooltip displayed (now has a title).

11. Magic Button problems during GoToMeeting sessions has been resolved. GoToMeeting poorly handled transparent windows and therefore obscured magic button when user was NOT running the Windows 7 basic theme. Though a GoToMeeting bug a design change was made to accommodate it since Citrix has been slow to fix.

12. Changed Marker Editor to handle new field called marker "Type". The values can be "Condition for RegWin Recognition", "Anchor for Path Optimization" and "Condition for Action Launch". The last one is a new feature which allows one or more markers to have delayed evaluation until after the magic button is clicked. In the situation where there is a great cost in evaluating a marker (multiple seconds or screen flicker), this feature allows you to defer that cost until after the button is clicked. Then, if evaluation fails, a standard message will be displayed to the user and the action will not be called.

13. Changes have been made to the logic which determines the way the Appspace Wizard names the output. The output appspace name now includes the XModelID, the Application Name and a random 4 digit numeric suffix.

14. A fix was made to the way RatchetX.Commander names its named-pipes which support the RatchetX.CommandLine.exe access. This bug was causing an unnecessary error message when running multiple instances of RX on one Windows Instance (Terminal Services, Citrix, etc...).

15. Encrypting and password protecting of Action Code is now possible by overriding a property in your Action = ActionRuntime.PasgswordForEdit. If you override this property and return a non-null value, that value will be required to view or edit the action code in the Action Editor. This value has no effect on the runtime properties - only design-time in Action Editor. When an action is password protected, its source is encrypted in the action XML.

16. Fixed File/Save Error which required use of File/Save As in certain situations.

17. Added advanced macro feature which supports sending keystrokes and mouse clicks to Citrix Windows running on local desktop.

18. Fixed Salesforce login failure resulting in "LOGIN FAILED" message when loading commander.


**Production Release of Ratchet-X Version 5.5.0.381**

Version: 5.5.0.381

Date: 01/20/2015

Notes: This version of Ratchet-X contains the following changes:
1. All asset files (appspaces, xmodels and actions), extensions are "associated" with Windows and therefore can be launched by double clicking the asset file from Windows Explorer or the desktop.

2. File Open and File Save dialog boxes have been retooled to accommodate upcoming changes due for release with Ratchet-X version 6. Current changes are cosmetic in nature only.

3. Action Pack installations available for installation from the Commander's Tools menu.

---

Production Release of Ratchet-X Version 5.5.0.362

Version: 5.5.0.362
Date: 09/18/2014

Notes: This version of Ratchet-X contains the following changes:

1. Added "Scanner" logging debugging feature.
2. Fixed Address Verification sample action bug.

---

Production Release of Ratchet-X Version 5.5.0.360

Version: 5.5.0.360
Date: 08/13/2014

Notes: This version of Ratchet-X contains the following changes:

1. Fixed a bug with Appspace Wizard Tooltip truncation of some text on high DPI machines. Change included a slight visual change to the tooltip displayed (now has a title).
2. Magic Button problems during GoToMeeting sessions has been resolved. GoToMeeting poorly handled transparent windows and therefore obscured the magic button when the user was NOT running the Windows 7 basic theme. A design change was made to accommodate the GoToMeeting bug since Citrix was not planning a quick fix.
3. Changes made to the logic which determines the way the Appspace Wizard names the output. The output appspace name now includes the XModelID, the Application Name and a random 4 digit numeric suffix.
4. A fix was made to the way RatchetX.Commander names its named-pipes which support the RatchetX.CommandLine.exe access. This bug was causing an unnecessary error message when running multiple instances of RX on one Windows Instance (Terminal Services, Citrix, etc...).
5. Encryption and password protection of Action Code is now possible by overriding a action property Action = ActionRuntime.PasswordForEdit. If this property is overridden this property and returns a non-null value, that value will be required to view or edit the action code
in the Action Editor. This value has no effect on the runtime properties - only design-time in Action Editor. When an action is password protected, its source is encrypted in the action XML.

**Production Release of Ratchet-X Version 5.5.0.301**

Version 5.5.0.301  
Date: 01/24/2014  
Notes: This version of Ratchet-X contains the following changes:  
1. To assist remote debugging, a new Option is now available in the Commander options dialog box called DebugOptions. This string contains either the value "None" or a collection of tokens (like "Abc" separated with commas). These tokens are not published and this feature is for use strictly by support personnel when attempting to isolate problems out in the field.
2. A magic button tracking issue has been resolved. This bug affected 64 bit Windows and IE10 or IE11 target application screens when the user had unchecked the Windows performance option; "Show contents of window while dragging". The bug would result in the magic button remaining rendered in its original location prior to the window being moved.

**Production Release of Ratchet-X - Version 5.5.0.293**

Version: 5.5.0.293  
Date: 01/19/2014 11:27 PM  
Notes: This version of Ratchet-X contains the following changes:  
1. Fixed critical bug in in Package Editor that resulted in parsing error when downloaded from production page.

**Production Release of Ratchet-X - Version 5.5.0274**  
Version: 5.5.0.274  
Date: 12/22/2013 2:45 PM  
Notes: This version of Ratchet-X contains the following changes:  
1. OCR engine and SCAR added.  
2. Regwin templates for Java and Chrome added (including new connectors to support the new templates).  
3. New JAB installer (v2.0.2).  
4. Compact mode added (SPL Distributor only feature).  
5. Labels in Commander Action and Appspace tabs have changed context menu item "Add..." to "Add/Register..." respectively.
6. 64 bit support (excluding text scrape and shortcuts).
7. Windows 8 Desktop Mode support.
Commander Options Reference

This reference describes the various configurable Ratchet-X options listed in the Commander's Options page. To access the Commander's Options page in Commander, click **Tools | Options...** When you do, you will be presented with the **Ratchet-X Options** grid. The following reference describes each option in detail.

### Compact Mode Group

The options in this group pertain to the mode within which Commander is running and the system features to which the user has access.

#### CompactModeOnStartUp

Option Choices:
- Yes, start compact
- No start normally

Commander can be run in either **standard mode** or **compact mode**. Standard mode allows users to see the full Commander runtime client and have access to all system features and samples. To configure Commander to run in standard mode, select "No, start normally". This is Commander's default setting. Compact mode is a condensed UI version of the runtime that allows users to see only a runtime html-based splash screen (that can be expanded upon via hyperlinks), and have the ability to run only the actions that were distributed as part of the install. Compact mode is used mostly by Ratchet-X Distribution Partners who distribute purpose configured versions of Ratchet-X or enterprise customers that want to distribute a "locked-down" version of Ratchet-X. To run Ratchet-X in compact mode, select the option "Yes, start compact".

#### HideLink

Option Choices:
- No, show link
- Yes, conceal link

When running Commander in compact mode, you can include a link on the html splash page that allows the user to switch to standard mode and have access to Commander's options. The user can do this by clicking the **Configure** link. If you want to show the Configure link, set this option to "No, show link". If you do not want the Configure link to show, select the option "Yes, conceal link".
LockPassword
Option Value:
• Unlock password value

If the Configure link is set to display on the compact mode splash screen (via the aforementioned HideLink option), you can attach a password challenge that must be met in order to allow the user to switch to standard mode. To set this property, you must enter the password challenge code.

Magic Button Group
The options in this group pertain to magic button's behavior at runtime.

ButtonStyle
Option Choices:
• Titlebar
• Top
• Right

Commander gives you the ability to select where you want the Ratchet-X magic button to appear with respect to your target application screen. You can elect to have a version of the button appear: in the Titlebar (positioned to the left of the left most button on the titlebar), on Top of the titlebar (positioned on the left side), or to the upper Right side of the application window. Note the appspace author may have explicitly defined where the magic button must be placed with respect to the application screen. If this is the case, this option will have no effect on that particular application screen.

Create Mode
Option Choices:
• Only If Actions Are Available
• Always

This option specifies whether or not the magic button will be rendered depending upon the presence of alerts. When set to Always, Commander will always render the magic button regardless of whether or not alerts exist for a given screen. If set to Only If Actions Available, the magic button will render only if alerts exist for a given screen.
**Override Mode**

Option Choices:
- Auto-Overrides
- Standard-Overrides

This option specifies how the magic button is rendered depending upon the presence of one alert or multiple alerts. If there is only one alert available and this option is set to Auto-Overrides, the magic button is rendered with the icon associated with the alert. Further, when you click the magic button, it will automatically execute the associated alert. If set to Standard-Overrides, the magic button is rendered as normal regardless of the number of alerts unless there are is a manual override defined.

**Overrides**

Option Value:
- Loads Magic Button Override Manager (displays the number of overrides defined)

Normally, when the magic button is clicked, the alert menu opens and alerts are displayed. However, you also have the ability to instruct Commander to associate one specific alert with the alert button click when a regwin is found. In other words, rather than having to drop the alert menu, you can execute the associated alert by merely clicking the magic button. You might want to do this for specific users who merely need to run the same action most or all of the time from a given screen but still want to keep the other, less frequently used alerts available if need be. The user can override the override by simultaneously clicking the magic button while pressing the key defined in the `SuppressKey` option.

Note: The magic button icon will change at runtime to reflect the override.

**Misc Group**

The options in this group pertain to various, unrelated Commander options.

**ButtonStyle**

Option Value:
- Number of minutes between Appspace Editor auto-saves
Appspace Editor is one of two Ratchet-X platform tool that allows you to create appspaces. Since appspaces can get rather complex and take some time to construct, Appspace Editor comes equipped with an auto-save feature so you can quickly and easily get your work back in case of power failure, or dare we say, unlikely system abend. AutoSaveInterval allows you to define, in minutes, the time interval in which Appspace Editor initiates an auto-save. The shortest AutoSaveInterval you can set is 1 minute. If you want to turn auto-save off, set the AutoSaveInterval to 0.

If the Appspace Editor abends or is closed prematurely before a save, the next time you load Appspace Editor, you'll be prompted to use the auto-saved version.

**AutoStart**

Option Choices:
- Yes, start with Windows
- No, start manually

Allows you to determine whether or not Ratchet-X is loaded as part of the Windows boot sequence. Setting this option to "Yes, start with Windows" means you want Ratchet-X to load as part of the Windows boot sequence. Selecting "No, start manually" means you do not want Ratchet-X to load as part of the Windows boot sequence and that you will load it manually when needed.

**CommandLineAccess**

Option Choices:
- Allow
- Don't allow

Ratchet-X allows you to execute actions via the command line. This option allows you to turn this ability on or off. Setting the option to "Allow" permits this instance of Commander to execute actions via the commander line. Setting the option to "Don't Allow" prohibits actions from being executed from the Commander line.

**DebugOptions**

This option is reserved for internal use. Its purpose is to have be used to generate extensive logging information to be used to help diagnose bugs and issues when encountered on user's desktops. The
logs will be stored in the user's AppUserData folder. Leave this value empty unless working with a support engineer to resolve an issue.

**MacroThrottle**

Option Choices:
- Slowest
- Slower
- Default
- Faster
- Fastest

Many appspaces make use of Ratchet-X macros in order to set a screen's state or prepare certain fields for extracts and pastes. The speed with which a macro executes is controlled by the appspace author. Since an appspace cannot be tested on every user's computer, it's possible for the appspace author to set macro execution speeds based on the performance he sees in his development and test environments. This means that the macro may need to execute more slowly or can be executed more quickly on a given user's computer. Macro throttling gives you the ability to speed up or slow down the performance of macros when executing on a given computer. If you find a macro is executing slowly or is executing too fast for your computer to handle, you can adjust the throttle accordingly to accommodate your computer. This option can be set to one of aforementioned five speed settings relative to the default speed (which is one of the five settings).

**NetCheck**

Option Choices:
- Check network at startup
- Don't check network

Checks to see if you have internet connectivity at Commander startup. "Check network at startup" means Commander will perform an internet connectivity check at start up. "Don't check network" bypasses the internet connectivity check. Since Ratchet-X performs a license check at load time, you should set this option to "Check network at startup" unless Ratchet-X is running in an offline environment which usually requires a special license.
ScannerInterval
Option Value:
• Commander scan interval defined in milliseconds (range 500 - 10000 milliseconds and defaults to 1500)

Defines the time interval, in milliseconds, Commander will use to scan the desktop when determining which applications you’re running.

StayOnTop
Option Choices:
• Yes, Stay On Top
• No, Float On Desktop

Determines whether or not Commander is the top-most application on the desktop. "Yes, Stay On Top" keeps Commander always as the top most application on the desktop. "No, Float On Desktop" instructs Commander to maintain its desktop load order with reference to other desktop applications.

SuppressKey
Option Value:
• Key value

Allows you to define a key to override the following Commander behaviors:
• Button overrides. Pressing the SuppressKey will block any defined MagicButton Overrides.
• When the alert window is displayed, pressing the suppress key keeps the alert window displayed within the associated regwin even if you click on another application window.
• Overrides the NetCheck value at load time by perfuming the opposite function defined in the NetCheck option.

Snaggle
The options in this group pertain to the behavior of snaggles.

SnaggleKey
Option Value:
• Defined snaggle key as selected in the subsequent Keystroke Selector dialog

Pops up a subordinate dialog (Snaggle Keystroke Selector - see below), that allows you to define the hot key combination that will activate any registered actions that contain a Start By Snaggle interface. More simply stated, it allows you to define the hotkey used to activate snaggles.

The snaggle keystroke selector allows you to define the keystroke that will be used to activate snaggles. To create a new keystroke, merely enter the keystroke as you would at runtime. As you type in the keystroke, it will be displayed in blue in the selector box. To save the keystroke, click the OK button. If you want to turn the keystroke off, check the No Snaggle Key checkbox and click the OK button.

**SnaggleOverrideMode**

Option Value:
• Auto-Override
• Standard-Override

SnaggleKeyOverrideMode allows you to define whether or not you're presented with the snaggle execution screen when there is only one snaggle available for a given application screen. When set to "Standard-Override", the snaggle activation screen will always be presented regardless of the number of snaggles available on the screen. When set to "Auto-Override", if there is only one snaggle available for an application screen, that snaggle will be auto executed upon activation.

**Sounds**

The options in this group pertain to Commander audio cues.

**MagicButtonAppear**
Option Choices:
- Sound On
- Sound Off

Specifies whether or not you want Commander to play an audio cue when the Ratchet-X magic button appears in a Ratchet-X enabled application.

**Paste**

Option Choices:
- Sound On
- Sound Off

Specifies whether or not you want Commander to play an audio cue when Commander pastes data into an integrated application screen field.
Sample Integrations

Ratchet-X ships with a number of sample applications and related assets. These applications and assets allow you to see Ratchet-X in action without having to build anything first. They also serve as examples of how to create assets that perform some of the primitive functions you'll need for most Ratchet-X projects (data extractions, pastes, mappings, etc). Although you should not need to work with the samples outside the context of Ratchet-X Commander, you can find all the samples in the `\Samples` folder located beneath the Ratchet-X install folder. By default, this folder can be found at `\Program Files (x86)\Ratchet-X Desktop Integration\Samples`.

**Sample Applications** - All sample applications can be loaded via Commander's **Tools** menu.
- MyCRM For Windows
- MyCRM For Web
- MyCRM For Legacy
- MGA Express

**Sample Appspaces**
- MyCRM Screen Simulator
- RatchetX.MGAExpress
- RatchetX.MyCRM
- RatchetX.MyCRMLegacy
- RatchetX.MyCRMWeb
- RatchetX.OtherSamples
- RatchetX.ScreenDataTransfer

**Sample Actions**
- RatchetX.AddressVerification
- RatchetX.DocToMGA
- RatchetX.Jump2MyCRM
- RatchetX.Jump2Web
- RatchetX.PhoneDialer
- RatchetX.QuickCopy
- RatchetX.RegWinDebugger
Since the samples are primarily distributed to demonstrate functionality, we will describe them all within the context of actions.

**Address Verification**
The address verification sample action allows you to verify a mapped US mailing address against the United States Postal Service's database. The action is actually a wrapper for a third party web service made available by SmartyStreets and using the RatchetSoft test account. This sample action can be executed against any of the MyCRM sample applications (MyCRM For Windows, MyCRM For Web or MyCRM For Legacy). Using MyCRM For Windows as an example, before you can execute this action, you need to first register with Commander the appspace **RatchetX.MyCRM** and the action **RatchetX.AddressVerification**. To execute the action, do the following:

- Load MyCRM For Windows sample application (keep the **People** tab active).
- Click the magic button to display the alert menu.
- Single click the alert entitled; **Verify Address: 1049 Park Avenue**.
- When the action runs, the address is extracted from the MyCRM screen and passed to the Address Verification action. The action then passes the address to the SmartyStreets web service where it is verified. The verified address is returned to the action's UI (small dialog in located at the bottom, right of the screen). All parts of the verified address that have been updated are rendered in green From the action UI, you can do the following:
  - highlight the fields into which the verified address will be pasted by clicking the **Highlight** link
  - paste the verified address data into the address fields in MyCRM by clicking the **Paste** link
  - close the action UI by clicking the **Close** link

This sample demonstrates data extractions; pastes, UI screenlink highlighting, use of the action UI and other general action protocols.

**RatchetX.DocToMGA**
The RatchetX.DocToMGA sample action demonstrates allows you to paste contents of a computer generated PDF file in multiple screens within a Windows application (**MGA Express**). Before you can execute the action, you must first register with Commander the appspace **RatchetX.MGAExpress**, and the action **RatchetX.DocToMGA**. To execute the action, do the following:

- Load MGA Express sample application (keep the **Insured** tab active).
• Load the sample PDF file **LossNotice1000414.PDF** located in the \Samples folder within the Ratchet-X install folder.

• The magic button should render in the PDF file's title bar.

• Click the magic button to display the alert menu.

• Single click the alert entitled; **Import Loss Notice into MGA Express.**

• When the action executes, it extracts the data from the PDF file and displays the extracted data in the actions UI window located at the bottom, right of the screen).

• To paste the extracted data to the appropriate screens in MGA Express, click the link **Paste this data to MGA Express.**

This sample demonstrates data extractions; pastes, multi-screen application navigation, use of the action UI and other general action protocols.

**RatchetX.Jump2MyCRM**

The RatchetX.Jump2MyCRM sample action demonstrates the ability to jump from a document to a record in the MyCRM For Windows sample application. Before you can execute this action, you must first register with Commander the appspaces **RatchetX.MyCRM,** **RatchetX.Jump2SharePointSample,** and the actions **RatchetX.Jump2SharePoint** and **RatchetX.Jump2MyCRM.** To execute the action, do the following:

• Load MyCRM For Windows (keep the People tab active).

• Click the magic button to display the alert menu.

• Single click the alert entitled; **Jump2SharePoint for Oscar Madiso.**

• When the action runs, it extracts the name **Oscar Madiso** from the First and Last fields, loads Internet Explorer and searches an instance of Sharepoint hosted by RatchetSoft for documents related to **Oscar Madiso.**

• In the list of documents, single click the first document named; **NOTICE OF CLAIM.** This will open the document in the current IE tab.

• Close the MyCRM application because we are going to demonstrate the ability to load MyCRM and navigate to the Oscar Madiso-related record.

• Click the magic button that resides in the title bar of the NOTICE OF CLAIM document. The action then loads MyCRM and performs the steps necessary to navigate to the Oscar Madiso-related record - just like the user would.
**RatchetX.Jump2Web**

The RatchetX.Jump2Web sample action allows you to load a URL while replacing specific parameters with data extracted from an application screen. Before you can execute this action, you must first register with Commander the appspace RatchetX.MyCRM and the action RatchetX.Jump2MyCRM. To execute this action, do the following:

- Load MyCRM For Windows (keep the People tab active).
- Click the magic button to display the alert menu.
- While you are free to click any of the Jump2 actions, for the purpose of this demonstration, click the alert entitled; Jump2 Google Maps for 1049 Park Ave.
- When the action runs, it extracts the address values from the address field, loads Internet Explorer, calls Google Maps and replaces the address parameters with the extracted address value. The result is the rendering of a Google map of the extracted address.

This sample demonstrates the ability to call a URL and replace the query string parameters with data extracted from a screen.

**RatchetX.PhoneDialer**

The RatchetX.PhoneDialer sample action allows you to extract a phone number from a screen and pass it to a phone dialer. This example uses both snaggles and the Ratchet-X dialer simulator so you don't have to install a dialer or set up an account with a provider. Before you can execute this action, you must first register with Commander the action RatchetX.PhoneDialer. Since this demonstration uses snaggles, there is no need to register a specific appspace. However, you will need to set up a snaggle key. To set up a snaggle key, click here.

Once the snaggle key is set up, do the following:

- Load MyCRM For Windows (keep the People tab active).
- Click the snaggle key you set up above. When you do, the snaggle will be executed. Note the three phone numbers on the screen will be highlighted.
- Click any of the highlighted phone numbers to sent the number to the dialer simulator.
- The simulator will load and simulate dialing the phone using the number you selected.

This sample demonstrates snaggles, extractions and shortcuts.

**RatchetX.QuickCopy**

The RatchetX.QuickCopy sample action demonstrates the ability to copy data from one application screen to another. Before you can execute this action, you must first register with Commander the
appspace **Ratchet.MyCRM** and the action **Ratchetx.QuickCopy**. To execute this action, do the following:

- Load MyCRM For Windows (keep the **People** tab active).
- Click the magic button to display the alert menu.
- Single click the alert entitled; **Copy Address to Company Tab**.
- The address information is extracted from the screen and you are prompted by the action UI (small dialog in located at the bottom, right of the screen) to click the **Company** tab.
- Click the MyCRM and the action will pastes the address information extracted from the People tab into the address fields on the Company tab.

This sample demonstrates extracts, pastes and action UI prompts.

**RatchetX.RegWinDebugger**

The RatchetX.RegWinDebugger action is more of a diagnostic tool than a sample application. The action can be executed from any application that is described in a registered appspace. In other words, you do not have to create a specific appspace that works with the RegWin Debugger. Think of this action as a quasi-universal action that allows you to do the following to any integrated screen: review regwin information, markers, snippets, clickits and mappings. Further, it also allows you to test extractions and pastes without having to write a specific action to process that data. For more on how to use the **RegWin Debugger, click here**. Before you can execute this action, you must first register with Commander the action **RatchetX.RegWinDebugger**. To execute this action, do the following:

- Load any application screen that has an appspace registered. For the purpose of this demonstration, if the **RatchetX.MyCRM** appspace is registered with Commander, load **MyCRM For Windows** and leave the **People** tab active.
- Click the magic button to display the alert menu.
- Single click the alert entitled; **Debug People Tab RegWin**.
- The debugger will load so you can review or test the regwin for this screen.
Licensing: Your license file and how it is used

Ratchet-X is licensed on a per installed seat basis. This means each workstation that has Ratchet-X Commander installed upon it must be covered under a valid license (either evaluation or full license).

**License Files**

Ratchet-X license files are digitally signed files stored centrally on the ratchetx.com server and downloaded to the user's machine during the activation process. When Ratchet-X is installed from our website, no license file is included. However, when Ratchet-X is run for the first time on a workstation, Ratchet-X prompts the user to enter licensing details for either a evaluation or a full license (licensee/activation). Typically, one Ratchet-X license file is generated for all users for a given organization. For example, if you have a Ratchet-X 20 user license, you're issued one license file (and corresponding activation code), which can be applied to each of the 20 installed seats.

**License Prompter**

The first time the users runs Ratchet-X, he/she is prompted with the Ratchet-X Licensing dialog (see Figure 1 below).

**Figure 1**

![Ratchet-X Licensing dialog](image-url)
**Evaluation Licensing**

When you download Ratchet-X from our website, you are sent an email which contains a link to the evaluation download location and an **evaluation license code**. After you install the software and run it for the first time, you are prompted to enter license information. If you are evaluating Ratchet-X, you should enter the evaluation license code in the **Evaluation Code** field in the Ratchet-X Licensing dialog when prompted. The evaluation license allows you to use Ratchet-X for 60 days without limitation. When that trial period ends, Ratchet-X will load until a full license is applied. You can, however, request an extension to a trial using the email address sales@ratchetsoft.com.

**Activation: Full Licensing**

When you purchase a full license of Ratchet-X, you are given both a licensee and activation code. When prompted for a license, you enter these codes by selecting the **Activate Now (Internet Required)** radio button. **Please make sure you enter these two codes exactly as provided.** Once your codes are confirmed (Internet connection required to confirm the license on our server), your license file will be retrieved from our server and copied to your machine. This file is refreshed each time Ratchet-X loads. This refresh allows us to easily apply any additional licensing features you may need. In rare cases where internet access is not available, RatchetSoft can supply a physical license file which can be copied to the local environment manually and referenced using the **Use local license file** radio button.

**Licensing and the Package Editor**

A very popular feature of Ratchet-X is the **Package Editor**. This tool allows an administrator to create a custom Windows installer file (MSI), which contains both the Ratchet-X software itself as well as all the assets and configuration data for each user. The package editor also allows you to enter your license information (see **Figure 2** below - **Licensee** and **Activation Code**), which will be automatically included in the windows installer package which, when installed, automatically activates the license and by passes the user license prompt.
Activating a License From Within Commander

In addition to being able to activate a license when Ratchet-X loads, you can also activate a license from within a running version of Commander. If you are running Ratchet-X under an evaluation license, you can activate a full license from the following two places depending on which mode Commander is running.

- If Commander is running in **Full or Normal Mode**, you can get to the Ratchet-X Licensing dialog and activate a full license by clicking **Help | Activate License** (see **Figure 3** below).

---

**Figure 2**

![Screenshot of License Activation Dialog]

**Figure 3**

![Screenshot of Commander Interface]

---
If Commander is running in **Compact Mode**, you can get to the Ratchet-X Licensing dialog and activate a full license by clicking the link **Activate License** link located at the bottom, left corner of the compact mode dialog (which is available by double clicking on the Commander icon in the Windows system tray (see **Figure 4** below).
Appspaces: What, How and Why?

Appspaces are the platform assets that contain information related to:

- how Commander reliably finds an application screen on the user's desktop
- how to determine if the found application screen is in the proper state to execute an action
- how screen UI elements map to xmodels, and by association, actions
- how to extract data from and paste data to screen fields

Physically, appspaces are XML files created with either the Appspace Editor or the Appspace Wizard and get registered for use with Commander via Commander's Appspaces tab. Appspaces are comprised of regwins and connectors. A Regwin, short for registered window, usually equates to one application screen you want to integrate with Ratchet-X. We say usually because it is possible to create a regwin that represents a portion of a screen or create multiple regwins for one specific application screen. It all depends on your specific use case and design.

Regwins are comprised of the following:

- Regwin header information
- Window selection information
- Markers
- Snippets
- ClickIts
- Mappings
- Shortcuts

**Regwin Header Information**

Regwin header information is used to identify and describe a regwin throughout the platform.

**Window Selection Information**

Window selection information is the first layer of application screen recognition. Window selection information includes: process name, class name and title bar text (which can be parsed using regex to accommodate your specific requirements). This is the kind of information you see when you review the Application and Processes tabs in Windows task manager for a given application.

**Markers**

A marker is screen link that serves one or both of the following purposes:
• supplement the Window Selection Information to help reliably identify an active regwin (usually to help determine application screen state)
• help optimize the regwin scanning process by serving as an alternative screen offset

Markers That Improve Regwin Identification Reliability
Sometimes the combination of process name, class name and title bar text with a regex validation is not enough to uniquely identify a regwin. This is often the case when an application uses the same text in the title bar for more than one application screen. For example, an application might embed the application name in the title bar of all its screens. If this is the case, you can define markers to further distinguish this screen from other screens that have the same characteristics. Markers are application and/or web page UI elements that are used to help distinguish one regwin from another. If you think of a regwin as a fingerprint, markers act as the set of distinguishing characteristics that, when taken collectively, help uniquely identify the regwin. For all intents and purposes, most UI elements such as fields, labels, graphics, title bar text, etc. can serve as markers.

Since it's not always apparent that the combination of process name, class name and title bar text will uniquely identify a regwin, it's a good practice to define at least one marker per regwin.

Markers That Improve Regwin Identification Reliability
Contrary to what you might think, the number of MSAA nodes that comprise the average application screen is not limited to the number of UI elements you visually see on a screen. In fact, MSAA is a very complex hierarchy of windows and objects that can number in the hundreds or thousands per application screen. Needless to say, anything that can be done when defining a regwin to reduce the number of nodes in the hierarchy which Ratchet-X has to process will yield runtime performance gains. Because of this, with a regwin, you can create markers that serve as offsets within the MSAA tree thus reducing the ultimate number of nodes Ratchet-X needs to inspect to find regwins and snippets. A good analogy to this concept is the following. Let’s say someone told you there was a bird in a tree outside your house. To find that bird, you could either start at the top or the bottom of the tree and traverse every branch looking for the bird. However, if you knew what branch the bird was on, you could dramatically reduce your search time by using that branch as your starting point. This same concept applies to Ratchet-X and the MSAA tree. If you define a marker that serves as an offset for a given group of nodes (this kind of marker is called an anchor), the time it takes to properly identify a regwin and gather all the screen elements it needs for extracts and pastes is dramatically reduced. This reduction in processing is more dramatic the more often you use the
anchor within the regwin. Sometimes the combination of process name, class name and title bar text with a regex validation is not enough to uniquely identify a regwin. This is often the case when an application uses the same text in the title bar for more than one application screen. For example, an application might embed the application name in the title bar of all its screens. If this is the case, you can define markers to further distinguish this screen from other screens that have the same characteristics. Markers are application and/or web page UI elements that are used to help distinguish one regwin from another. If you think of a regwin as a fingerprint, markers act as the set of distinguishing characteristics that, when taken collectively, help uniquely identify the regwin.

For all intents and purposes, most UI elements such as fields, labels, graphics, title bar text, etc. can serve as markers.

**Snippets**

Snippets, working in tandem with Connectors and XModels, is the mechanism by which application screen fields and other UI elements are exposed to the Ratchet-X platform for data extractions and pastes. As mentioned above, **Connectors** are also part of regwins.

**ClickIts**

ClickIts are UI elements that are defined for the purpose of facilitating application navigation and event initiation. Technically, they are application screen nodes just like markers and snippets. However, they do not contain data that helps to uniquely identify a screen, support extracts or support pastes. Rather, they are nodes that support mouse clicking to help set an application screen in the proper state for processing. ClickIts are also used to attach the execution of an action to a UI element (such as a button or graphic click, or a hot key from a text field). ClickIts are used primarily from within action snippets, working in tandem with Connectors and XModels, is the mechanism by which application screen fields and other UI elements are exposed to the Ratchet-X platform for data extractions and pastes. As mentioned above, **Connectors** are also part of regwins.

**Mappings**

Mappings are how you link screen data to an action and action results data back to application screens. Mappings are facilitated via a platform construct called xmodels.

**Shortcuts**

Shortcuts are keyboard hot key combinations or mouse click combinations that when activated,
execute actions. Think of a shortcut as a key or mouse activated magic button override. Shortcuts allow you to execute actions without having to go through the magic button's alert window menu. Shortcuts can be configured to be either globally available from within a regwin, or local to a specific UI element. For example, if you want to set up a shortcut that allows a user to access images from a document management system using screen information as the search criteria and attach that to a hotkey combination, you can do so. This configuration will allow the user to initiate the document search regardless of where the cursor is situated on the screen (thus yielding the same results the shortcut is activated). However, let's say you want to allow the user to perform a Google Maps lookup based on a screen address and there are multiple addresses on the screen. In this case, you can set up a shortcut that renders the map using the address that resides in the field where the cursor is when the user activates the shortcut. Local shortcuts add an additional layer of context to the shortcut.

Connectors
Connectors are pieces of code that facilitate communications between specific UI element types (text field, combo box, radio button, etc), and Ratchet-X. Connectors contain the logic that allow data to be extracted from and pasted to application screen fields. While connectors are created and managed independently of regwins, they are utilized within regwins when they are applied to snippets. Ratchet-X comes configured with connectors that can communicate with most Windows and browser application UI element types. However, if the need arises, you can also create your own custom connectors.
Templates: What are they and how do I use them? Can I create my own?

All regwins within an appspace, be that appspace created via the Appspace Editor or Wizard, are based on templates. Although templates are used to create new regwins within an appspace, a template is actually a special purpose appspace that contains: one, and only one, regwin; an optional marker that may be needed to target that regwin type (i.e. Windows application, Internet Explorer page, etc), or establish the point within the regwin from which all screenlink searches must start; and the connectors needed to communicate with the screenlink types present within the regwin.

Where Do I Come Across Templates?

When using the Appspace Editor, you come in contact with templates upon creating a new regwin (see Figure 1 below). The New RegWin dialog requires you to select a template that best matches the regwin type you're creating. Ratchet-X currently ships with the following templates:

- Standard Windows application (Default). Used for all 32 and 64 bit Windows application screens.
- Internet Explorer web page.
- Google Chrome web page.
- Java Client (requires JAB V2.0.2). JAB stands for Java Accessibility Bridge. The JAB makes a java-based application's UI accessible in a manner similar though not exact, to MSAA (Microsoft Accessibility API).

Although it's usually obvious as to which template works best, sometimes it may require a little trial and error to determine which template is best equipped to handle your specific regwin.
Figure 1

When using the Appspace Wizard, you come in contact with templates when you select your target Window (see Figure 2 below). Unlike the Appspace Editor, the Appspace Wizard automatically selects for you the template it thinks will work best based on the screenlink types it finds on the screen. We'll get into this in more detail below.

Figure 2
Where Are Templates Physically Stored?
The templates that ship with Ratchet-X are stored in the Templates folder located beneath the Ratchet-X install folder. By default, this folder can be found at Program Files (x86)\Ratchet-X Desktop Integration\Templates. In order for Ratchet-X to differentiate an appspace file from a template file, the template file must be named with the extension .appspacetemplate.

Can I Create My Own Templates?
Absolutely. In order to create your own template, you can load the Appspace Editor and create one from scratch. However, the more common way templates are created is by starting with an existing appspace and then paring it back to just the components you need (regwin, marker, connectors).

When paring back an existing appspace, you need to be aware of the following rules:
- Templates must contain one, and only one, regwin.
- The Description value is displayed whenever this template is referenced in the Appspace Editor and Wizard so be sure you include a concise yet informative description.
- Window selection parameters (i.e. Process Name, Class Name and Titlebar text parameters), should be as broad as possible since this information is highly subject to change in production. Use regex liberally.
- Templates may contain one marker. This marker usually serves as the anchor for the main window (i.e. the offset point for all screenlink navigations).
- Templates usually do not contain snippets, clickits, shortcuts or mappings. However, if for any reason you include any of these in your template, these items will be carried over to any appspace in which the template is used.
- You can have as many connectors in your template as you need and name them whatever you want. However, there is a very important naming consideration if you want the Appspace Wizard to be able to automatically select this template when analyzing a screen. If you want the Appspace Wizard to be able to select this template, the name of your connector must end with either a valid MSAA role type or the Ratchet-X catch-all connector type, "Scrape". This is important because the connector type designation is how the Appspace Wizard determines which connectors it needs when analyzing a screen. For example, when naming a connector that extracts text from a field label in a Windows application screen, the connector name must end with .StaticText. You are free to use whatever naming convention you want before the ",", but after the dot, you need to use a valid MSAA role type or Scrape.
• Templates need to be saved with the `.appspacetemplate` extension or else Ratchet-X will not recognize the file as a template.

• Once the template is created, it can be used in the following three ways:

• In the Appspace Editor, you can manually open the template file and use it as the starting point to create your template. Just remember to do the following two things. First, the **Browse For Appspace** dialog defaults to file type `.appspace` so templates will not be displayed. In order to show template files, you need to change the file type selection value from **Appspace files (*.appspace)** to **Template files (*.appspacetemplate)**. Second, when you save your appspace, remember to select **Save As** instead of **Save** so you don't save over your template. Make sure the file extension is `.appspace`.

• When you create a new regwin, the **New Regwin** dialog asks the question; "**Which of the following best describes your target screen?**" In order to get your template in the subsequent drop down, you need to place your template file in the `\Templates` folder and manually update the `appspacetemplate.xml` which also resides in the same folder. You need to add the following item to the file:

  `<TemplateItem>
      <Name>YourTemplateName.appspacetemplate</Name>
  </TemplateItem>`

**Note:** The order of the items in this file is important because that is the order the Appspace Wizard will use when analyzing a screen and selecting a template. When analyzing a screen, the Appspace Wizard will select the first template it finds in this list that appears to be a match for the selected window.

• Optionally, you can provide the user with supplemental information regarding the use of your template. This information is provided to the user in the form of a link. For example, the Java template that ships with Ratchet-X requires a minimum version of the Java Accessibility Bridge (JAB), in order to function properly. The link to download the JAB is displayed to the user in the following two places within the Appspace Editor and Wizard respectively (see **Figures 3 and 4**).
Figure 3

Figure 4
In order to add such a link to your template, you must manually add the following lines to your `.appspacetemplate` file:

```xml
<TemplateMessage>Insert the link text to be displayed</TemplateMessage>
<TemplateMessageURL>Insert link</TemplateMessageURL>
```

Finally, you should be aware of the following processing rules with regard to the automatic selection of templates and connectors within the Appspace Wizard. First, if the Wizard does not find a match with a high degree of certainty, it will select the Standard Windows application template by default. Second, if the Wizard does not find a connector whose post "." text matches exactly the MSAA role of the screenlink being assessed, it will select the Scrape connector by default.
XModels: How can I make my action independent of my target screen?

An XModel is a data schema, or microformat, that describes some collection of information related in some way. For example, the data contained in a business credit report could serve as an XModel. The collection of data contained in the report is considered the XModel and the individual data elements are called XModel elements. The purpose of an XModel is to associate snippets with a commonly understood semantic (i.e. meaning). We use the phrase "commonly understood", because any snippet that supports a specific XModel can be processed by an action that knows how to process that same XModel. This association of snippets and actions through an abstraction layer is what allows Ratchet-X to scale in the way it does. In other words, since snippets and actions map to XModels, no snippet is explicitly aware of a specific action and no action is aware of a specific snippet or application. Think of XModels as a late binding mechanism for snippet definition. For example, a zip code can be processed by external applications for many different purposes. If you want to "expose" a given zip code snippet in an application for use by an action, you can do so by mapping the UI element that contains the zip code to a snippet and then mapping that snippet to a zip code XModel. By doing so, any existing or future action that supports the zip code XModel will be able to process that application’s zip code. That’s very powerful.

Physically, an XModel is an XML file that contains elements and various configuration parameters that define the meaning and supported capabilities of the XModel. XModels are defined using the Ratchet-X XModel Editor.

XModels are mapped to snippets via the Mappings tab when editing a regwin in the Appspace Editor.

XModel Editor

The XModel Editor is the Ratchet-X tool that allows you as an action developer to define XModels. The XModel editor can be loaded from Commander’s Tools menu. When the XModel editor loads, you have the option of either creating a new XModel or editing an existing XModel. To create a new XModel, click File | New. Creating and configuring an XModel is a simple process. All you have to supply is a name, define its elements and configure its capabilities. Let’s take a closer look at the XModel Editor screen.

ID

Like all asset and object IDs within the Ratchet-X platform, XModels must be identified by an ID which can consist of any combination of letters and/or numbers - no spaces or punctuation marks.
Description
To help others understand the data your XModel describes, we recommend you provide a detailed description of your XModel. Remember, the more people that use an XModel that you support, the more people can use your actions and appspaces.

Elements / Element Detail
Elements is the section of the screen where you define the XModel elements. To add an element, click the Add button. When you do, a default element will be created with the name XModelElement followed by a sequential number (the first one added will be named XmodelElement1). The element is added to the list on the left and the element configuration details resides in the Element Detail section to the right. Once the element is created, you should change its ID to something more descriptive of the element and supply a detailed description.

Next, piece of information you need to supply is the element’s extract/paste capability. An element can be configured to support either full extract and paste capability or extract only. Select the capability you want to apply to your XModel element from the Capability drop down.

Finally, you may also supply an Initial Constant value which will be assigned to the xmodel element any time it is referenced within the mapping section of the appspace editor. This value can be overwritten or re-mapped within the appspace editor. This capability is useful when you want to default a URL or location value in your appspace.

Once you’ve configured your XModel, you can either click the Add button to add additional elements or you can click File | Save As to save your XModel.

The only other features available on the screen is the ability to reorder elements within an XModel. Keep in mind that appspace creators will see the elements in the appspace editor’s mapping tab ordered the way you define the order here so make sure you order your elements in a logical fashion. For example, if you’re defining an XModel that contains a US address, it’s a good idea to group the address elements together and order them in accordance with the way address data is usually ordered (address line 1, address line 2, city, states, zip code, country).
Paths: How we find screen fields

One of the biggest challenges faced when building any kind of Windows automation is consistently finding a given screen field. Application users see a field with a label (e.g. an Invoice Number), preceding it. Unfortunately, the Windows operating system does not provide any "unique key" to consistently identify that field in the future. Usually, the only thing the OS provides is a window handle which changes each time the screen is loaded. When we built the Ratchet-X platform, this was our first challenge. Without a reliable way to target a specific screen field, the rest of what we offer would be useless. Our solution to this challenge is a technique we call "pathing". Before we delve into the details of pathing, let's first cover a few things about how Windows organizes and exposes elements of an application's user interface.

Background

Back in ancient times (maybe 1999 or so), Microsoft introduced a new feature to the Windows operating system called "Microsoft Active Accessibility" or MSAA for short. This new feature was designed to pave the way for screen readers for the visually disabled. The goal of MSAA was to have establish an API within the OS that allowed every Windows desktop application to present its user interface elements (textboxes, labels, combo boxes, etc), as a series of nested objects resembling the layout of the application as visually seen by the user. This "tree" of elements starts with a root element (the main window), and has containers which subdivide the screen into sections and, within each section, have other containers or fields. Every application is different and there are really no rules about how this "tree" needs to be formed. Most developers have no idea that their application's user interface is made available via this API and make no effort to support it. However, whether they know it or not, they do support MSAA because they often use third-party controls to construct their applications which provide rudimentary support. Out in the real-world we have seen all shapes and sizes of these UI trees. Some are very shallow with all the fields existing as first level children of the root. Others are much more hierarchical with many nested containers stretching as far as the eye can see, with all the fields clustered at the very end of these branches. Most, however have a normal depth of about 5 or 6 container levels before the fields appear. It really doesn't matter, we can handle them all.

In Windows jargon, each element of the tree (from the root to the farthest leaf), is called an IAccessible object and each has its own properties describing the element and its state. These tree elements provide us all the information we need to perform our automation tasks. For the purpose of integrating applications through Ratchet-X, the most important properties are:

- **Name**: the name of the element. This is often empty because the author (developer), did not bother to set it. But with certain element types (StaticText), this property does have a Value and it is the data you see on-screen.
- **Role**: an element type which defines the behavior of that field such as Window, PushButton, ComboBox, Textbox, StaticText, RadioButton, Checkbox, Link, Graphic, Scrollbar. Refer to the MSAA Wikipedia page for a complete list.
- **Description**: the description of the element. This is almost always blank.
• **State**: though presented as one property, it really is a bitwise collection of states that can be on or off. Some common states include Invisible, ReadOnly, Focused, Available, Offscreen, Checked and Selected.

• **Value**: the string value of the element. In a properly designed Windows application, you use this property to read and write the "value" of a Text or Combobox element.

• **Children**: a list of other tree elements which are nested beneath the element you are reviewing.

• **Parent**: a tree element which acts as a container for the element you are reviewing. If the current element is the root node of your application, then its parent is the Windows desktop itself.

• **Action**

• **Windows Class**

To summarize, the Windows OS allows each application to be remotely inspected and manipulated by another process running in the same Windows desktop. Although its original purpose was to enable screen-readers and other assistive technologies, windows accessibility has opened the door for other automation tasks such as Ratchet-X actions.

**The Path**

Generic Path Syntax

is: (Anchor)/Axis:/Role[Condition]/Axis:/Role[Condition]/Axis:/Role[Condition]/Axis:/Role[Condition]

**Anchor** can be either (Root) or (Marker=Anchor)

**Axis** can be Child, Parent, Left, Right, Up, Down, LogicalNext, LogicalPrev, SiblingNext or SiblingPrev. If Axis is not explicitly provided, the default is child axis.

**Condition** The following syntax defines how to define a condition: [Property Operator 'Value']

**Property** is can be any of the valid MSAA properties listed above.

If the property you are including in your condition is **Role**, you must set the role comparison value to any of the valid MSAA roles listed in the table below (note you can use the "*" wildcard to denote any valid MSAA role:  

---

RATCHETSOFT KNOWLEDGE BASE – SERIES 6
If the property is a **Name, Value, Description, Action** or **WindowClass**, you can set your condition to any string value.

If the property is **State**, must set the state comparison value to any of the valid MSAA state listed in the table below.

#### Table: Valid MSAA State

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
<th>Action</th>
<th>WindowClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertHigh</td>
<td>AlertMedium</td>
<td>Animated</td>
<td>Busy</td>
</tr>
<tr>
<td>Checked</td>
<td>Collapsed</td>
<td>Default</td>
<td>Expanded</td>
</tr>
<tr>
<td>Floating</td>
<td>Focused</td>
<td>HasPopup</td>
<td>HotTracked</td>
</tr>
<tr>
<td>Indeterminate</td>
<td>Invisible</td>
<td>Marqueed</td>
<td>Mixed</td>
</tr>
<tr>
<td>Moveable</td>
<td>None</td>
<td>Offscreen</td>
<td>Pressed</td>
</tr>
<tr>
<td>Protected</td>
<td>ReadOnly</td>
<td>Selected</td>
<td>SelfVoicing</td>
</tr>
<tr>
<td>Sizeable</td>
<td>Unavailable</td>
<td>Valid</td>
<td>Visible</td>
</tr>
</tbody>
</table>

**Conditional Operators** are "=" for equal, "!=" for not equal and "~" for matches regular expression.

**Conditional Value** must be enclosed in single quotes.

**NOTE:** By default, paths utilize a node's ordinal position (it's position for a given role type within a branch of the MSAA tree). If you want to address a specific ordinal position, you can refer to it as Role[Index='n'], where
"n" is the index value, or you can use the shorthand version of [n]. For example, with regard to referring to a node's ordinal position, Window[0] is equivalent to Window[Index="0"].

**Axes**
Valid axes are:

(Heirarchy) Child, Parent  
(Spacial) Up, Down, Left, Right  
(Tab Order) LogicalNext, LogicalPrev  
(Index Order) SiblingNext, SiblingPrev

**Sample paths**
The best way to get up to speed on pathing is to look at real world example. The table below contains a list of common pathing examples along with a brief commentary for each.

<table>
<thead>
<tr>
<th>Sample Path</th>
<th>Commentary</th>
</tr>
</thead>
</table>
| (Root)/Window[0]/Text[0]                         | Simple path on child axis  
Starting at the root node, find its first child node which has a Window role. Then find that node's first child that has a Text role. Note: 0 means first, 1 means second, etc... (offset by zero) |
| (Root)/Window[1]/Client[0]/Window[3]/StaticText[22] | Simple path - but deeper  
Starting at the root node, find it's second child node which has a Window role. Then find that node's first child node that has a Client role. Then find that node's fourth child node that has a Window role. Then find that node's twenty third node which has a StaticText role. |
| (Root)/Window[1]/Client[0]/Window[3]/StaticText[Name='First Name'] | Non Ordinal Condition  
Similar to preceding but instead of using the twenty-third child text node, we inspect all the StaticText nodes at that level of the node tree and return the first which matches the condition Name='First Name'. |
| (Root)/Window[1]/Client[0]/Window/StaticText[Name='First Name'] | Single Role Wildcard  
Similar to preceding but introduces a subtle single-role "wildcard" behavior. The path will start at the Root node, then to the second child Window, then the first child Client, then, since no condition is specified (notice no square brackets), all Window nodes will be searched, in child index order. Eventually the processing will be performed on the fourth Window (as in previous) which does have a matching StaticText node which will be returned. This "wildcard" behaviour will introduce some extra searching and should only be used when needed. |
Any Role Wildcard
Similar to preceding, but introduces the "any role" symbol of asterisk. When used instead of a specific role, the asterisk will instruct the search to inspect all nodes on that axis regardless of role type. In this case, the StaticText node will be returned if no prior nodes have a Name property matching "First Name".

Full Wildcard
Similar to preceding but instead of defining each level of the tree hierarchy, a "wildcard" is used (expressed as double slash "/"). This begins an exhaustive search of all descendents (child, grandchild, etc.), attempting to match the remaining path item(s). In this case, the processing would start at the Root node, then it would begin inspecting each descendent node searching for StaticText nodes with the Name property matching "First Name". This path may return the same node as prior path example, but it will be slower. It's more flexible but it's not as efficient because it will definitely involve more node inspections to complete. Wildcards are quite useful when the location of a node may vary, but use judiciously due to the performance implications.

Changing Axis
In this path, we do a wildcard ("//") search starting at the root node for a StaticText node with matching Name property "First Name". But when we find it, we switch to the logical next axis and find the third node. Specifically, this path reads as "Starting from root, do a full wildcard search through the child axis looking for a StaticText node with a name matching "First Name". Once found, switch to the logical next axis. Then find the first node on that axis. Then find the next. Then find the next. Note: the asterisk is used here to specify "any role" and since no condition is present (no square brackets), each level just moves to the next node. Note: Logical next is determined by the application's tab order and is not available on all nodes.

Axis Conditionals
Similar to the prior path, this path finds the first StaticText node with a matching name property, but then it switches to the logical next axis and searches for the first Text node (in tab order), with a value property that matches 'Justin'.

Multiple Axes
Similar to the prior path, this path finds the First Name node and switches from default child axis to its reverse the parent axis. Once the axis is changed, it finds the first node (first parent), then switches axis again, this time to the left axis. Now it moves to the third node along the left axis. There is no limit to the number of axis changes which can be made.

Anchors
This path does not start from the root node. Instead, it starts with an anchor node. An anchor node is a marker created for the purpose of easy (short), navigation to multiple nodes. This path reads; starting from the MainAnchor node, find the eighth StaticText node. Using anchors is a great way to improve performance when you have an application with a deep nesting of fields. For example, you may find your target snippets nested at a level ten levels down from the root node. For efficiency, you may create an anchor which contains the common path leading to the deepest nested container to all your nodes which may be at level eight or nine. Then, when you create your other paths, you start at the anchor and the paths you create are only one or two levels deep and the logic to find that anchor node is performed only once for all your subordinate paths.

SiblingNext
In this path, we start at an anchor node and do a wildcard search ("//") on the default child axis to find a node with a matching name property. Once found, we switch to the sibling next axis and move one node. Sibling next and
previous moves from child to child in the node tree. For example if we found the SSN node and it happened to be fifth child under its parent node, this code would move one node to the sixth child.

```
[Marker=MainAnchor]//StaticText[Name~'Field\d&State!='Invisible']
```

**Compound Condition - Regex operator & State Property**

In this path, we start at an anchor node and do a wildcard search ("//") on the default child axis to find a node that matches a compound condition. First, the name property must match the regular expression (tilde operator) "Field\d" for example Field1, Field2, etc. Second the node must not be in an invisible state. Compound condition can use "&" to represent "And" and "|" to represent "Or". Highly complex conditions involving multiple nested And/Or statements are not currently supported.

Note: due to the need to use escape sequences on special characters like carriage returns or tabs, regex patterns that require the use of the slash should use double slash as shown in this example.

```
[Marker=MainAnchor]/Window[0]/(/Window[Name='Zip Code'])(/Window[Name='Postal Code'])(/Window[Name='Postal District'])/Text[0]
```

**Alternation (If/Else) Logic**

In this path, we start at the MainAnchor and search down the default child axis for first Window. Now, the parser encounters an open parenthesis which begins an alternation. An alternation uses parentheses to define alternate paths. In this example, there are three. Each alternate (from left to right) will be searched and once a match is found any remaining alternates will be skipped and processing will pickup after the last alternate segment. This is a very useful feature when you have dynamic screens which have slightly different (but consistent), layouts and you don't want the overhead of a full wildcard search.

```
(Root)/Window[0]/PageTabList[0]/(/#Version9.1#/PageTab[3])(/#Version9.2#/PageTab[2])
```

**Inline comments & Alternation (If/Else) Logic**

In this path, an alternation is used and inline comments are added. Inline comments are added as individual segments separated by slash and must be enclosed in hash for example /#COMMENT#/ . The trailing dash is only required if additional path segments will be added.

```
(Root)/Client[0]/Window[3]/:PixelOffset(y='45',x='0',dpix='120',dpiy='120')[Role='Pane'&Name='SKU']/StaticText[0]
```

**PixelOffset function**

In this path we navigate normally to a window node then we perform a "PixelOffset" function. This specialized function uses the current node as a starting point to jump a specified number of pixels to land on a new node. This jump is calculated using the top left of the starting node and adding the "x" and "y" values supplied as parameters. That new destination is queried to determine the active accessibility node found there and it becomes the next node in the search.

Note: The runtime environment’s DPI is an issue when using PixelOffset. Different desktops may be using a different DPI and therefore the target values may require adjustment to account for it. To simplify this, we provide dpix and dpiy parameters which allow you to define the DPI of your encoding environment where you know it works. Then, at runtime, if the DPI is not the same as your encoding environment, the X and Y values will be corrected to make up for the difference in DPI. If DPI parameters are not explicitly provided, the assumption is standard DPI of 96.

```
(Root)/Window[3]//Text[Name='ID']/#NOCACHE#
```

**NOCACHE Directive**

In this path we perform a wildcard search for a Text node using a name condition. Trailing after the text node condition is a special comment called "NOCACHE" which signals the run-time engine to not cache this node. Without caching, this node will require a search every time its used, even though the RegWin may appear to be alive. This is useful when you are working with quirky applications that have an internal organization that is very dynamic and UI elements are retired and replaced. This should be used cautiously because it has a negative effect on performance.

```
(Root)/Window[3]/#P:AltRect(x='10',y='0',w='40',h='16',dpix='120',dpiy='120')#
```

**Alternate Rectangle Directive**

In this path we find the target node in the usual manner. What makes this path different is that it ends with a special comment called AltRect which allows us to define a location (screen coordinates), which is separate from those of the target node. This "alternate rectangle" will be used at runtime as if it's the boundaries of the node instead of the
actual boundaries. These values are encoded using an offset which is relative to the left top of the actual node. So in this example, our offset is 10 pixels to the right of the target node's left top and its got a width of 40 and a height of 16. Our encoding machine was using a higher resolution so the DPI was encoded here just as it is in PixelOffset. If DPI parameters are skipped, its presumed to be standard DPI of 96. Alternate rectangles are used primarily to defines zones for TextScrape and OCR.

(Marker=SomeAnchor)/Window[0]/Client[0]/:GoTo(Name='Top')

**GoTo function**

In this path, normal navigation occurs to the Client node. Then a special function is used to navigate to the top level desktop window. This function is used rarely, but it becomes necessary when you need to navigate to the top window and the number of levels of the path are not fixed.
Magic Button configuration: Titlebar, top, right and custom placement

The primary way Ratchet-X actions are executed is via the magic button (click here for information regarding alternative methods for executing actions). Since the magic button is the most common way to execute an action, Ratchet-X offers a significant amount of flexibility with regard to where that button can be placed within a target window. By default, the magic button is placed within the title bar to the left of the left most button (which is usually the minimize button). While this default placement often suffices, sometimes placing the magic button in the title bar can be problematic (e.g. too many buttons on the titlebar, title bar button conflict, or the titlebar is not actually a "Windows" generated title bar and doesn't adhere to standard behavior). If you need to place the magic button elsewhere, you have the option to display it either on top of the title bar or to the right of the targeted window. These options are available to you as the appspace author within the Appspace Editor via the Window tab and clicking the Button Configuration link. Clicking this link will bring up the Magic Button Configuration dialog (see Figure 1 below).

Figure 1

Allowable Button Locations Group

The Allowable Button Locations group serves the following two purposes:

1. Defines where the button may and may not appear.
2. Defines the explicit position of the title bar, top and right side locations.

Via the Ratchet-X Options ButtonStyle, the user is allowed to set the default location of the magic button in his instance of Commander. However, this parameter will only take effect provided the
underlying regwin permits the magic button to be rendered in that position. If the appspace author unchecks any of the locations in this group, the magic button will not be allowed to be rendered in that position regardless of the user's preference. So, if all the boxes are checked (the default setting), the magic button will render in the title bar unless the user has a different preference set in Ratchet-X Options. If any of the boxes are unchecked, the magic button will be rendered based on the user's preference assuming that preference coincides with an allowable rendering position within the regwin.

The X Offset and Y Offset values are set to "nudge" the magic button the defined number of pixels from the 0,0 position for the setting. These settings are as follows:

**On Titlebar of target window**

![On Titlebar of target window](image)

Magic button is placed to the left of the left most button on the title bar.

**On Top of target window**

![On Top of target window](image)

By default, the magic button is placed on top and 20 pixels to the right of the upper left corner of the target window.

**On Right side of target window**

![On Right side of target window](image)

By default, magic button is placed on the on the outer right side and down 30 pixels from the upper right corner of the target window.
If you want to see the effect of your settings, click each of the **Test** links and Ratchet-X will render the button for you in the corresponding location.

**Advanced Group**
The Advanced Group contains the following magic button configuration options:
- Place button on Main Module Window
- Force polling to assure button visibility

**Place button on Main Module Window**
When a main module window spawns a child window, you may still want to (or in some cases, have to as is in the case where the child window does not have a title bar), place the magic button on the main module window instead of the child window. Checking this option instructs Commander to always render the magic button on the main module window.

**Force polling to assure button visibility**
Since Ratchet-X integrates with target windows from the outside-in (i.e. the target application window is not explicitly aware of Ratchet-X or the magic button), Ratchet-X must "hook" the window at the operating system level in order for the magic button to render within and track the window. However, since Ratchet-X Commander is a native 32 bit application, it cannot hook 64 bit windows thus preventing the magic button from properly rendering in the target window's title bar. This being the case, the way to mimic the effect of hooking is to check the **Force polling to assure button visibility** checkbox. This will force Commander to spy the target window and re-render the magic button when needed. While there are certain side effects related to this method of tracking the window, the only effect you may notice is that when you physically move the target window, the magic button will not re-render until the completion of the next scanner interval cycle after the window move is complete.

Aside from native 64 bit applications, you may come across some 32 bit applications that present window hooking consistency problems (though rare). As a general rule, if the magic button is rendering consistently and you're sure you're not having regwin recognition issues, you should turn this option on.

For more information regarding magic button display issues, [click here](#).
Conditional Markers vs. Anchor Markers

Markers come in two different types: conditional and anchor. This topic will explain the difference.

**Conditional markers** serve the purpose of uniquely identifying a RegWin. Typically, you add one or more conditional markers to a RegWin when the process, class name and title bar text are not distinct enough to identify the desired state of the application. Using MyCRM as an example, you may need to create a conditional marker to ensure that the "People" tab is selected. This conditional marker would confirm that the page tab ScreenLink with the Name "People" also has a "Selected" state. This is the primary and most obvious use of markers. There is another use for markers, however, that is equally important but often overlooked. Sometimes we need to create markers which are used to optimize performance by minimizing the number of steps involved in locating all the elements you will access on a target RegWin. We call these "Anchor Markers" or simply "Anchors".

**Anchors** are simply markers that are used by other ScreenLinks (Snippets/Markers/ClickIts) as the starting point of a path instead of the standard "(Root)" element. In the examples below, you will see standard paths starting from the Root and stretching down many levels to find a specific Text node. In each of these cases, Ratchet-X needs to navigate down from the root and inspect 10 node levels to find the Text node. So to resolve all these nodes will require about 50 level inspections.

(Root)/Window[0]/Client[0]/Window[3]/Client[0]/Window[0]/PageTabList[0]/PageTab[3]/Client[0]/Window[0]/Text[Name='ID']
(Root)/Window[0]/Client[0]/Window[3]/Client[0]/Window[0]/PageTabList[0]/PageTab[3]/Client[0]/Window[0]/Text[Name='Name']
(Root)/Window[0]/Client[0]/Window[3]/Client[0]/Window[0]/PageTabList[0]/PageTab[3]/Client[0]/Window[0]/Text[Name='Date']
(Root)/Window[0]/Client[0]/Window[3]/Client[0]/Window[0]/PageTabList[0]/PageTab[3]/Client[0]/Window[0]/Text[Name='Start Date']
(Root)/Window[0]/Client[0]/Window[3]/Client[0]/Window[0]/PageTabList[0]/PageTab[3]/Client[0]/Window[0]/Text[Name='End Date']

But you will notice that these fields are all clustered together (very common) and share a common path for the first 9 levels. To improve performance you can create a marker whose path leads to
that shared ancestor node and can be referenced by the five snippets above. Each of those snippet paths can start its search at this anchor node instead of the default "(Root)" node so the path syntax can be simplified and Ratchet-X performance improved due to the removal of redundant steps in path resolution. This example shows the creation of an anchor marker and its usage in other snippets.

MainAnchor Marker

\[= \text{(Root)/Window[0]/Client[0]/Window[0]/PageTabList[0]/PageTab[3]/Client[0]/Window[0]} \\]

Then we define the snippets starting at that anchor point to reduce the amount of searching required to resolve our 5 snippets.

\[\text{(Marker->MainAnchor)/Text[Name='ID']}\]
\[\text{(Marker->MainAnchor)/Text[Name='Name']}\]
\[\text{(Marker->MainAnchor)/Text[Name='Date']}\]
\[\text{(Marker->MainAnchor)/Text[Name='Start Date']}\]
\[\text{(Marker->MainAnchor)/Text[Name='End Date']}\]

In this case, the number of search levels went from 50 to 14. We resolve MainAnchor using 9 searches (levels) and 5 searches for our snippets (1 search each snippet). This efficiency dramatically increases the speed of path resolution making your Ratchet-X action seem noticeably quicker. But speed is not the only benefit. When you use anchors, you can often make your Appspace easier to maintain. Look at the example above again. You will notice that the anchor path includes "/PageTab[3]/". If the target application supports user drag drop of tabs, reliance on ordinal index position may not work. By having this path declared only once, it only needs to be changed in one location. We can change the anchor path to include "PageTab[Name='Student']" and it will be automatically applied to all dependent ScreenLinks.

Contrary to this topic's title, conditional markers and anchor makers are not mutually exclusive. You can create a marker which is both a conditional and anchor. That's because an anchor marker is simply any marker which is used/referenced as an anchor in another ScreenLink's path. It's possible that a specific ScreenLink may contain information used as a unique identifier for your RegWin and be a container holding all your fields. In this case, the same marker would be used as a condition and be referenced as an anchor. These overlaps are rare though. Normally, the conditional markers
are visible items with identifying text in the Name/Value property and the anchor markers are simply container nodes with no visual characteristics. So when you create a marker which is anchor-only and not needed as a condition, you should uncheck the "Required for RegWin Recognition" option in the marker editor to avoid unnecessary processing during RegWin recognition.
Connectors: Custom coding inside your appspace

A connector is a small block of C# code used to add unique processing to your appspace. When you create a new RegWin, you start-off with some default connectors specific to your target application (Standard/IE/Chrome/Java) but you can write your own. Created using the Appspace Editor, connectors are a "coding" experience and require some knowledge of C# and .NET. Armed with that knowledge, you should find creating connectors easy. First, connectors require you to create a class which subclasses "ConnectorRuntime" - a built-in type inside Ratchet-X. See example below.

```csharp
using System;
using System.Windows.Forms;
using RatchetX.Scripting.Connectors;

namespace Standard
{
    public class TextToUpper : ConnectorRuntime
    {

        public override string Extract(ScreenLink screenLink)
        {
            return screenLink.Value.ToUpper();
        }

        public override void Paste(ScreenLink screenLink, string data)
        {
        }
    }
}
```
screenLink.Value = data;
}
}

In this example, the connector is called Standard.TextToUpper and implements two methods Extract and Paste. In this case, the Extract method returns the Value property of the snippet's ScreenLink and forces it to upper case. The Paste method simply assigns the supplied string data to the snippet's Value property.
Scrape and OCR: When accessibility fails you

The primary technique Ratchet-X uses to extract data from application screens is MSAA querying. However, certain application types are poor MSAA clients and do not provide the requisite information required to extract information. In these cases, you should try one of the following techniques:

- TextScrape
- OCR

**TextScrape Technique**

TextScrape uses the windows API to "hook" a target application, request a repaint of the desired zone and then monitor the application's text drawing commands. This method is very reliable and works for most applications. If you've created a regwin and tried to use the MSAA extraction technique without success, you can try this method by applying the appropriate Scrape connector. Alternatively, if you want to get a quick "lay of the land" evaluation of your application screen's ability to be scraped in this fashion, you can perform test using the Scrape test tool which can be found in the Appspace Editor's Tools menu. This method allows you to test your screen without having to create a regwin.

**OCR**

When all other extraction techniques fail, screen OCR is your last good hope. We say "good" hope because we've tweaked our screen OCR engine to perform quite competently without the need for an additional font training file (though our professional services group can create one for you if need be). The following section describes how to configure the OCR connector.

Screen OCR can be used to extract data from a zone by applying to a specified snippet, the .OCR connector found in all Ratchet-X shipped templates. Once you've selected the snippet zone and applied the connector, you then need to click the Parameters link which in turn brings up the Connector Properties property sheet (see Figure 1 below).
By far, the OCR settings parameter is the most comprehensive and important parameter. When you click in the Settings property, an ellipsis button will be rendered. When you click that button, you will be presented with the SCAR (Screen Capture And Recognition), Editor (see Figure 2 below).
Figure 2

Screen Capture and Recognition

Screen Capture

Capture Area | Color Filter | Blob Filter | Zoom

Fine-tune zone by nudging its size/location

Offset Left: 0, Offset Width: 0
Offset Top: 0, Offset Height: 0

Reset

✓ Turn Off Font Smoothing

Recapture

Text Recognition

Filter | Training | Hints | Advanced Vars

Select one character filter

None

OCR Result | Status | 0000 ms |

Recognize

OK

Cancel
The SCAR Editor screen is divided into two sections, each containing its own set of tabbed subsections. The top section is the **Screen Capture** section. The Screen Capture section is the place you define and process the actual zone image you defined as your snippet. The second section is the **Text Recognition** section. The Text Recognition section is the place where you manipulate the text extracted from the image you defined and processed in the Screen Capture section.

**Screen Capture Section**

The Screen Capture section is divided into the following four subsections:

- Capture Area
- Color Filter
- Blob Filter
- Zoom

**Capture Area**

When you define a zone for your regwin using the Snippet Editor, the image is extracted and displayed in the window below the Recapture button (image window). The default size displayed is the "fit to window" sizing. If you hover over the image window, you can toggle this setting to display the processed image (processed means any color or blob filters and zoom factors are applied). By default, the only processing effect applied is a zoom factor of 2.7.

**Offset Controllers**

The Capture Area also has a set of nudge controls that allow you to fine-tune the size and location of your snippet zone. Since snippets are defined by fixed points within the application screen, SCAR does not allow you to modify the originally captured coordinates. Rather, it allows you to define offset values that will be used to alter the region of the zone you want processed. **Offset Left** offsets the left margin of the image. **Offset Top** offsets the top margin of the image. **Offset Width** offsets the right margin of the image. **Offset Height** offsets the bottom margin of the image. Clicking the **Reset** link resets the offset controller values to 0, or no offsets.

**Turn Off Font Smoothing**

In order to make text more readable to humans, Windows has a feature called **Smooth Edges of Screen Fonts**. While this feature is nice for humans, it interferes with screen OCR. This being the case, every time Ratchet-X performs screen OCR, it checks to see if this feature is turned on or off at the operating system level. If this feature is turned on, Ratchet-X turns it off and performs OCR. If the **Turn Off Font**
Smoothing checkbox is checked, after preforming the OCR, Ratchet-X will leave the feature turned off. If not checked, Ratchet-X will perform the OCR and turn the feature back on. Ratchet-X's manipulation of this feature lasts only for the current Windows session. When the user reboots, the Smooth Edges of Screen Fonts feature is set back to whatever the user's Windows setting is. By default, the Turn Off Font Smoothing checkbox is checked.

Recapture
When you define a regwin zone, Ratchet-X is actually making a copy of the image zone and passing it to the SCAR. The SCAR does not maintain a live connection to the target window so if you change data in the target window, those changes are not reflected in the SCAR unless you click the Recapture button.

Color Filter
Colors extraneous to the main font color cause noise for the OCR engine and reduces recognition rates. On the Color Filter tab, you have the opportunity to drop out the colors that are not germane to the text you are trying to OCR. You can drop out a color by clicking the color(s) you want to drop. An "X" rendered over the color square denotes the color will be excluded from processing. As you manipulate the color filter, note the effect of your action in the image window. The more noise you can exclude, the more accurate the recognition results will be.

Blob Filter
In order for the OCR engine to decipher where one letter ends and another begins or to differentiate characters from lines and extraneous markings, the OCR engine performs something called blob extraction. On the Blob Filter tab, you have the ability to modify the width and height settings that define a blob in your image. When you click the Use Blob Filter check box and manipulate the width and height sliders, notice the lower values define smaller acceptable blob sizes, thus making the OCR engine more exclusive in its process. Setting the sliders to higher values makes the acceptable blob size larger resulting in a more inclusive process. You should modify these values until you get just the characters you want processed (as best you can). This feature is good for dropping out field borders, check boxes and other graphical objects. At any point, you can click the Show Blobs link to see the borders of your defined blobs.

OCR recognition results improve when a line of text is lined-up on the same horizontal line. Unfortunately, programmers are notorious for inconsistent alignment in their applications. When you check the Nudge blobs into horizontal alignment checkbox, SCAR will do it’s best to line up objects close in proximity, to the same horizontal line.
Zoom
OCR recognition rates improve when the image is zoomed. The zoom tab allows you to set the zoom factor for image processing. The default setting for zoom is 2.7 since it usually yields the best results. If 2.7 does not work well, try 3.4. Beyond that, it becomes a matter of trial and error to get the best results.

Text Recognition Section
The Text Recognition section is divided into the following four subsections:
- Filter
- Training
- Hints
- Advanced Vars

Filter
The Filter section allows you to supply various text filter criteria to the OCR engine to help improve recognition. Note you can only supply one filter type at a time.

Training
The Training section allows you to select which training files you want the OCR engine to use to perform OCR. Currently, Ratchet-X ships with the following two training files: eng and engCommonScreenFonts. Since these two files contain complementary instructions, you should use both training files at all times.

Hints
Hints allows you to provide the OCR engine some insight into the nature of the strings it is OCRing. These insights pertain to patterns and spacing. The "Don't bother with plausibility" option instructs the OCR engine to not guess at a value based on linguistic logic. This option should be turned on when dealing with mix alpha and numeric code values. For example, if this option is turned on and your data contains the code "4GE", the OCR will ignore its desire to recognize the string as the word "AGE". The "Don't trust spaces less than this" option allows you to define a space factor that determines how the OCR engine inserts spaces into its output strings. The default is 1.6 and should suffice in most cases.
Advanced Vars

Ratchet-X uses the Google Tesseract engine for OCR services. Though the SCAR allows for the manipulation of some engine processing parameters, there are literally hundreds of others that we have not included since they did not seem applicable to Ratchet-X. However, if you want to configure a parameter that is not accounted for in SCAR, you can do so by manually entering the variable and value pair in this section. Note: If you use this feature...good luck because you're on your own!

If you want to see the results of your work at any time during the process, click the Recognize button and the results will be displayed at the button of the screen in the OCR Results tab.

One final point worth noting is OCR processing speed is directly proportional to the size of the zone defined. So the rule of thumb is to select as small a zone as possible to maximize performance.
Scanner Log: Which Regwins are found and why some are not found?
The Scanner Log Viewer (SLV), is a diagnostic tool that helps you diagnose regwin recognition problems. The SLV provides you with a scan-by-scan logging of the regwins sought (based on the appspaces you’ve enabled), regwins found and the reason for a regwin recognition failure. The most common reason regwins fail is because a marker that is configured to assist with recognition is not found. When you load the SLV, it will look like the screen depicted in Figure 1 below.

Figure 1

The Scan Sessions panel on the left lists the time of the scan associated with the log entry. For example, if you have Commander configured to scan every 1.5 seconds, you’ll see an entry placed into this list every 1.5 seconds. The number of regwins found is displayed within parenthesis to the right of the entry. Each time the number of regwins found either increases or decreases, an asterisk will be placed to the left of the entry making it easier for you to identify when the change in the active regwin recognition state occurred. For example, at 6:00:23PM, Commander’s active regwin recognition state changed from finding no active regwins, to finding one active regwin.

The Scan Sessions panel on the left lists the time of the scan associated with the log entry. For example, if you have Commander configured to scan every 1.5 seconds, you'll see an entry placed into this list every 1.5 seconds. The number of regwins found is displayed within parenthesis to the right of the entry. Each time the number of regwins found either increases or decreases, an asterisk will be placed to the left of the entry making it easier for you to identify when the change in the active regwin recognition state occurred. For example, at 6:00:23PM, Commander’s active regwin recognition state changed from finding no active regwins, to finding one active regwin.
The **RegWin Details** panel to the right lists all the active regwin recognition details associated with the selected scan session. All found active regwin entries are rendered in green and flagged with an asterisk. If a regwin is found, the entry will tell you the number of markers found for the given regwin. If a regwin is not found, the entry will tell you why the regwin failed to be recognized. While in many cases the reason for failure reported will actually be the reason why a regwin was not found, sometimes the reason for failure reported may merely be a symptom of the actual problem.

Remember, the SLV is a diagnostic tool designed to help you figure out the problem, not definitively state the problem or solve the problem.

The SLV adds new scan entries to the bottom of the scan sessions list. If you check the **Auto-select** check box, the scan session entry selected will always be the newest entry added to the list. However, if you uncheck the box, the entry you currently have selected will remain selected regardless of the number of new scan sessions added to the list.

**Why Might Some Regwins Not Found?**

There are a number of reasons why a regwin might not be found. These include:

- More than one instance of a given regwin is open. In this case, Commander will recognize only the first instance it finds.
- The integrated application loads each time with a different class name and thus does not match the **Class Name** value you defined in the **Appspace Editor's Window** tab. In this case, you need to regex out the variable portion of the class name or leave the class name value blank.
- The integrated application’s title bar text contains dynamic information that changes from record-to-record. In this case, you need to regex out the variable portion of the title bar text.
- The regwin is not in the proper state per one or more markers defined.
- The integrated application screen is sized in such a way that the defined marker is not accessible.
Magic Button: Why is it not showing?

There are a number of reasons why the magic button may not be appear in a target application. Some of the reasons are obvious, some not. The following details issues that impact the rendering of the magic button in your target application.

Is Ratchet-X installed and running?

Don't laugh. As users become more accustomed to cloud-based solutions, the expectation that applications are "magically" available to them is becoming more pervasive. If a magic button is not appearing as expected in a target application, make sure Ratchet-X is indeed installed and running in the Windows system tray.

Are the required assets registered?

In order for Ratchet-X to recognize a target window, it must be referenced by a regwin contained within a registered appspace, and that regwin must be mapped to a registered action. If a magic button is not appearing as expected in a target application and Ratchet-X is installed and running, make sure the required assets (appspace and action), are registered in the appropriate tabs in Commander. If you're in the process of creating or modifying those assets, make sure that you're working with the correct version of your assets when testing. When you create an appspace using the Appspace Wizard, upon save, the appspace is automatically registered with Commander. However, if you're creating or modifying an appspace or action using the Appspace or Action Editor, the assets are NOT automatically registered or updated within Commander. This is why when you close the Appspace or Action Editor and Commander reloads, you are asked the question; "Do you want to reload actions and appspaces?" It's generally a good idea to click "Yes", if you've made any changes to registered assets.

Do you currently have a development tool loaded?

When you load any of the Ratchet-X development tools (Appspace Editor, Action Editor, XModel Editor, Package Editor or Appspace Wizard), Commander is paused. You can tell Commander's run state by looking at the Ratchet-X icon in the system tray. If the standard Commander icon has a "pause" symbol super imposed over it ( pictureBox1 ), Commander is paused and not scanning for target applications. This will cause the magic button to not appear in a targeted application.

Assuming there are no load or state issues preventing the magic button from displaying, you should move on and investigate the following factors.
Check to see if Commander is finding the regwin

With the targeted regwin active on the desktop, click on Commander's Appspace tab and see if there is a green circle with a white check mark (✓), to the right of the appspace name. If there is, that means that at least one of the regwins contained within that appspace is being found. It may not be the one you're looking for, but it's a good start. Next, double click the on the appspace. This will bring up the **Appspace Details** dialog. If you look at the **RegWins** label, you should see a listing of all the regwins contained within the appspace. All found regwins are preceded by an asterisk. **Figure 1** below is an example of the Appspace Details dialog for the MyCRM sample application that ships with Ratchet-X when loaded and navigated to the People tab.

**Figure 1**

![Appspace Details](image)

In this example, note Commander has found three regwins within the MyCRM appspace (MainWindow, PeopleTab and TabStrip). If you don't see an asterisk next to the regwin you are targeting, you should then consider the following.

Do you have two instances of the same regwin loaded?

When Commander scans for regwins, it recognizes only one instance of a running regwin and that instance just happens to be the first instance it come across during the scan. If a magic button is not appearing as expected in a target application, make sure you do not have multiple instances of the regwin loaded. If you do, the magic button is probably being rendered in one of the other loaded instances.
Do you have a magic button setting conflict?
There are several settings within Ratchet-X that impact the rendering of the magic button and may be the cause of your rendering issue.

- **Commander Option: MagicButton | CreateMode** - When set to **Only If Actions Available**, Commander will not render a magic button in a target window that has no action (alert) available to run. Example conditions that might trigger this option include:
  - you do not have a mapped action registered in Commander.
  - a mapped action requires certain pieces of data that are not available in the regwin for the current record (for example, the address verification action requires an address line 1 value in order to show the magic button).
  - In the Appspace Editor on the Window tab, you have either the **Disabled** setting "checked", or the **Show Button** setting "unchecked".
  - In the [Appspace Editor | Button Configuration](https://example.com) dialog, you have the magic button configured to appear somewhere other than the title bar or your offsets are set at levels that are pushing the button off the target window screen.
  - In the [Appspace Editor | Button Configuration](https://example.com) dialog, you have the **Advanced** setting **Place button on Main Module Window** checked and the button may be appearing on the target window's parent window. This is common when the target window doesn't have a title bar or a "simulated" Windows title bar. Though not directly tied to this setting, there's a related condition that may also impact the appearance of the magic button. Sometimes a screen or dialog appears to be a subordinate window, but in reality, it's actually a user control on the main form disguised to look like a subordinate window. If this is the case, the magic button will appear on the main window's titlebar and not on the visual treatment of the user control.

And then there's the odd stuff
The following section describes some of the odd, but thankfully rare, magic button rendering issues.

- Target application is a running in 64 bit mode and the button appears under the target window.
- Scanner has stopped scanning for some reason.
- An action abended and has corrupted either the scanner or memory.
- Magic button is rendering at position 0,0 (upper left corner), of the desktop.
In the rare cases when these conditions have been reported, it's usually during the development process. We recommend you unload all Ratchet-X tooling, Commander and the target application and restart. That should clear the error condition.

**Nope, none of the conditions described above apply. Now what?**

If you've exhausted all the aforementioned possibilities, chances are, your regwin is broken. Why might your regwin be broken?

- The target application screen has changed to the point that screen nodes are not where they are expected to be.
- The application is in a state that the regwin author did not anticipate nor configure (this is a good thing to catch during testing).
- The regwin was configured to narrowly or specifically.

The first thing you need to do is check the scanner log. With the scanner log loaded, load and unload your targeted regwin noting the changes recorded in the scanner log. Concern yourself primarily with the items flagged with an asterisk, as the asterisk denotes changes in scan state. The scanner log also provides you some detail as to when and why a regwin is found or lost.

The next thing to do is load the Appspace Editor where you can perform a series of tests on your appspace to determine what is not working. The first set of tests are performed from the **Window** tab. With the target application loaded and the appspace loaded in the editor, you should cycle through the tests in the following order:

- **Window Only** - This test will check to see if you anything has changed with regard to the application Class Name, Process Name or Titlebar Text. If the test fails, note the error message and check the item in error. Usually, the source of the problem is one of the aforementioned test items needs to have a regular expression applied or, if one is already applied, it may need to be tweaked to accommodate the current state. Things to watch out for in this area include:
  - Class names with numbers in them. You may need to apply a regular expression.
  - Titlebar text that contains dynamic information such as record data. Again, applying an regular expression may be the solution.
- **Window With Markers** - This test collectively tests the Window and any defined markers. Assuming the Window Only test passed, a fail on the Window With Markers test indicates there's a problem with either an anchor (which will cause everything linked to the anchor to fail as well),
or a specific marker validation. Note which marker is causing the error and perform a path check as described below.

**Path Check**
In the Appspace Editor, locate the marker in error in the **Markers** tab grid. **Right click** and **Disable** the maker. Note the marker you just disabled appears in the grid with a strikethrough font. Then, **right click** again on the marker and select **Compare Path**. This will pop the **Path Comparison Tool** (see **Figure 2** below).

**Figure 2**

As you can see, the path to the broken marker is displayed in the **Existing** field. The next thing you need to do is re-select the marker in error by selecting the appropriate anchor and selecting the marker node using the **Crosshair Selector** (just as you did when you originally created the marker). When you do, the correct path will be displayed in the **Selected** field. Any differences between the original marker path (which is now broken), the correct path will be rendered in **red** (see **Figure 3** below).

**Figure 3**
The most likely reason for a broken marker is that your initial path definition relied too heavily on ordinal position of the node. When ordinal position of a node is unreliable, you should consider using other node targeting techniques.

If there is no issue with the path, then check to make sure your marker validation condition is still valid.
Java: Yep, we have that covered too

Ratchet-X supports Java-based clients in the same way it supports Windows applications - through an accessibility layer. However, Java applications do not natively support MSAA the way Windows application do. To achieve this, Java application users must install the [Java Accessibility Bridge (JAB)](http://ratchetx.com/java/JABSetup202.msi). The JAB allows Ratchet-X to access data contained in Java application screens for Java versions 5, 6 and 7 for both 32 and 64 bit implementations. The following describes how to install the JAB, how to test it and some important update considerations.

**Installing The JAB**

RatchetSoft has created an install file that makes installing the JAB easy. The install is available at [http://ratchetx.com/java/JABSetup202.msi](http://ratchetx.com/java/JABSetup202.msi). In addition, the install is also available as a link in the Appspace Editor on the New RegWin dialog (see Figure 1).

![Figure 1](image)

The JAB install is also available as a link on the Select Target Window dialog in the Appspace Wizard. This link is made available when a Java Client template is selected by the wizard after you have targeted a Java application screen.
When you download the JAB installer, the installer does the following:

- Installs the JAB software.
- Searches your computer for all installed versions of Java and updates each configuration with JAB-related information. We call this JAB registration.
- Installs a Program Group named **Java Access Bridge 2.0.2**. The following programs are placed in that group:
  - Java Monkey 32 bit - accessibility explorer used for testing
  - Java Monkey 64 bit
  - MyCRM for Java sample application
  - Rerun Java Access Bridge Registration - application that allows you to re-register JAB if need be (we'll get into why this may need to be done)
  - View Registration - view the registration log file

**JAB Install Considerations**

- The JAB installer should be run on every computer where Ratchet-X needs to integrate with Java-based client applications.
- All Java applications and Commander must be closed before the JAB installer is run.
- Only Java installed instances at the time the Java installer is run will be registered.
• Important - If after installing the JAB, the computer is updated or has a new instance of Java installed with a version of Java below version 7 update 6, the JAB must be re-registered after the update or install (this is why you would need to rerun Java Access Bridge Registration).

Java Monkey Accessibility Explorer
If you want to ensure the JAB is reporting back the accessibility information needed for Ratchet-X to work, you can test it by running the Java Monkey application. Assuming the JAB has been installed properly and is detected by Commander, to test with Java Monkey, do the following:
• Load the Java application you want to test. You can also use the MyCRM for Java sample application which resides in the same program group as Java Monkey.
• Load Java Monkey (either 32 or 64 bit depending upon your configuration).
• In the main window, you should see a tree structure the describes the accessibility structure of all running Java applications. If you do not see the structure, click File | Refresh. Once the structure is displayed, you can click through the tree and view the accessible elements available for each Java application.

Figure 3 shows the Java Monkey displaying the accessible nodes within MyCRM for Java.
You can always check to see if Ratchet-X is detecting the JAB by checking the About box in Commander or checking the Status tab if running Ratchet-X in Compact mode. If the JAB is not detected or it is detected but your Java application is not returning accessible information, just rerun the Java Access Bridge Registration. Running the Java Access Bridge Registration multiple times on a machine will not break anything.
Start By Scenarios: Is there an alternative to the Magic Button?
The most common way for a Ratchet-X action to be executed is via the magic button. The user navigates to a record in his line of business application, Commander detects the user is on a screen for which it has been trained to recognize and makes its presence known by placing the magic button in the application’s title bar. The user then clicks the button and executes the action. However, this is not the only way an action can be executed. For example, you may want to execute an action from a command-line prompt, based on the presence of a file in folder or a snaggle on a screen, etc. Ratchet-X offers you the flexibility to execute an execution in the manner most appropriate for your use case. These different methods for executing actions are called **Start By scenarios**. **Figure 1** below lists the StartBy scenarios Ratchet-X supports.

**Figure 1**
This is the list presented in the **Action Editor** when you first create an action. You're prompted to select the StartBy scenario you want to use by selecting the related action template. While you can only select one StartBy scenario, you can add additional StartBy scenarios either by manually referencing them in your code or selecting them from the Action Editor Toolbar.

Once you've selected your action template, the action editor will populate the code editor section with the template's action code. **Figure 2** shows the starting state of an action when the **Start by RegWin** template is selected.

**Figure 2**
Notice this action is implementing two interfaces: `iStartByRegWin` and `iStartByTest`. Action Editor requires you always support the `iStartByTest` interface because it’s used to test your code directly from the Appspace Editor itself. Each interface is contained in a code region and contains only default data. You always need to replace this template data with the actual data you’ll be using.

Let’s review each `iStartBy` scenario; its interface, how it’s coded what it does.

**StartBy Test Scenario (Figure 3)**

```java
//region StartByTest: test starting point - this is called only when run by Action Editor.

public ActionResult StartByTest(ActionContext context, ActionTestSimulator simulator)
{
    // defines parameter value to simulate what Commander manages for each user
    context.SetParameterValue("UserName","test1");
    context.SetParameterValue("Password","pw1");

    // Locate the appspace file to use for this test
    AppspaceInfo appspace = context.GetAppspace("installFolder://MyCRM.appspace");

    // Reference the RegWin we are targeting
    RegWinInfo regWin = appspace.GetRegWin("PeopleTab");

    // Make sure that RegWin is currently running on the desktop
    if (regWin.IsAvailable==false)
        throw new Exception("Testing requires that MyCRM be loaded and on People tab");

    // Call the main script which does all the real work
    return MainScript(context, regWin);
}

@endregion
```

**StartByRegWin: started by magic button or shortcut and related to a RegWin**

```java
package ActionResult StartByRegWin(ActionContext context, RegWinInfo regWin)
{
    return MainScript(context, regWin);
}
```

This interface is used by Appspace Editor as a starting point when the action author clicks the Run button in the editor’s tool bar (see Figure 4).
The **StartByTest method** is the starting point of the action execution in this scenario and the rest is up to you. We suggest as a best practice, to use this method simply to set up the test and call single method common to all your iStartBy interfaces. In this example, the StartByTest method is assigning some test login values to "action properties", locating the appspace file, referencing a specific RegWin, ensuring it’s currently running, and passing the RegWin into the MainScript where you will do all the actual work. This example is a typical example of an iStartByTest implementation when paired with an iStartByRegWin implementation. In **Figure 3** above, you see this example is primarily using a StartBy scenario and, at runtime, will receive a reference to the RegWin which hosted the magic button. In this case, our StartByTest method attempts to simulate this condition.

Now that we’ve coded this interface, we simply click the Run button and the StartByTest method is called directly and your action executes.

### Start By RegWin Scenario (Figure 5)

```java
#region StartByRegWin: started by magic button or shortcut and related to a RegWin

public ActionResult StartByRegWin(ActionContext context, RegWinInfo regWin)
{
    return MainScript(context, regWin);
}

#endregion

// Used by Commander to determine triggering RegWins. Returns an array of full RegWin keys
public string[] RequiredRegWins
{
    return new string[] { "http://ratchetx.com/appspaces/MyCRK/PeopleTab" };

#endregion

// ConfigureButton - Customize the appearance of the magic button alert
public override void ConfigureButton(ButtonContext context)
{
    context.Caption = this.Caption;
    context.HighlightIDs = context.GetVisibleIDs();
    context.Show = true;
}

#endregion
```
This is the most common StartBy scenario. It's used when the action author wants to work with a specific RegWin(s) by name. By using this interface, the action author is tightly binding the action to one or more specific RegWins and referencing it, or them, by name. The StartByRegWin method is the starting point for this scenario. It's used in code to manipulate all of the Ratchet-X platform and is essential at all points within your action code. It's a best practice to either pass it through all your method calls or store it in a class level property so it can be available everywhere in your code. The **RegWinInfo** object supplied as a parameter is how you access the information about the specific RegWin that hosted the clicked magic button. In **Figure 6** below, you can see the RegWinInfo object must be the MyCRM PeopleTab RegWin and it's safe to assume it's currently running or else the user would not have been able to click the magic button.

**Figure 6**
The **RequiredRegWins** property is used to define which RegWins will be connected by the magic button. Typically, there is only one. However, from time-to-time we find examples where multiple RegWins can be starting points for an action. This condition is supported by supplying multiple strings in this property. Each string is the URI which uniquely identifies the RegWin to which you're connecting. This URI, also known as the **RegWin Key**, can be found in the RegWin definition inside the Appspace Editor (again, see Figure 6 above). The RegWin Key creates the linkage between the appspace and action Ratchet-X Commander uses to determine when a button is placed and what actions are available on that button. The **ConfigureButton** method, (shown in Figure 5 above), is used to customize certain aspects of the magic button. You can change the button's icon, caption, hilights and visibility. The implementation shown is standard. For more details on changing the magic button's appearance see [Magic Button Configuration](#).

### Start By Mapping Scenario (Figure 7)

```csharp
#region StartByMapping: started by magic button or shortcut and related to a specific onscreen mapping (xmodel instance)

    public ActionResult StartByMapping(ActionContext context, MappingInfo mapping)
    {
        return MainScript(context, mapping); // return from Menu selection
    }

    // RequiredMappings: Used by Commander to determine triggering Mappings. Returns an array of XModel URIs
    public string[] RequiredMappings
    {
        get
        {
            return new string[] {
                                "http://mystuff.com/xmodels/JumpToPO.xmodel",
                                "http://mystuff.com/xmodels/JumpToInvoice.xmodel"
                              };
        }
    }

    // ConfigureButton - Customise the appearance of the magic button direct
    public override void ConfigureButton(ButtonContext context)
    {
        // If we are doing a Jump to PO
        if (context.GetAssociatedKey().Contains("JumpToPO"))
        {
            context.Caption = "Jump to PO"; // custom caption
            context.Icon = new Icon(context.GetEmbeddedFile("JumpToPO.ico")); // custom icon
            context.Hilights = context.GetVisibleIDs();
            context.Show = true;
        }
        else if (context.GetAssociatedKey().Contains("JumpToInvoice"))
        {
            context.Caption = "Jump to Invoice"; // custom caption
            context.Icon = new Icon(context.GetEmbeddedFile("JumpToInvoice.ico")); // custom icon
            context.Hilights = context.GetVisibleIDs();
            context.Show = true;
        }
    }

#endregion
```
This is a very common start by scenario. It's used when the action author wants to de-couple an action from any specific RegWin and allow for a different author (e.g. appspace author), to define his own RegWin which can be used with this action. This loose-coupling is achieved by using the XModel Editor to define an XModel which can be mapped to any RegWin using the Appspace Editor. Actions designed this way have a one-to-many potential thus allowing actions to have greater reach.

The StartByMapping method is called after the user clicks a magic button placed on any RegWin which has a mapping of one of your required mappings. The mapping is an indirect way of accessing the underlying RegWin for extract or paste. The appspace author, by mapping your XModel to this RegWin, provides you a source of data for each of your XModel elements. Without any prior knowledge of this appspace, you can now safely extract key fields for use in your action. Subsequently, others can map your XModel and get their appspace to work with your action. The StartByMapping method is supplied two arguments; ActionContext object and MappingInfo object. The ActionContext object is supplied to every start by method and was explained in detail above. The MappingInfo object is supplied as an argument and is a reference to the specific mapping inside a specific RegWin, inside a specific Appspace. Often, you don't care about the specific appspace, everything you need is accessible directly from the MappingInfo object. Its main use is to access the individual MappingElementInfo objects which support the Extract and Paste methods. These objects provide indirect access to the RegWin Snippets that do all the work.

The RequiredMappings property contains one or more strings which represent the URL location of the XModel(s) you'll be supporting. Typically, they are URLs starting with "http" or "assetfolder" - representing shared file locations. At runtime, Ratchet-X Commander acts as a broker, comparing the XModel URLs mapped in found RegWins (see Figure 8), to XModel URLs returned by this property and, if matching, your action will be available on the magic button.
The ConfigureButton method allows you to customize the magic button caption, icon, highlights and visibility. If you look at Figure 7 above, you'll see this method is applying a custom caption and icon based on the XMModel mapped. For more details on changing the magic button appearance see Magic Button Configuration. The IStartByTest interface required for testing within the Action Editor typically contains some code that collects the items that would otherwise be available when the IStartByMapping method is called, namely the MappingInfo object. See Figure 9 for a sample of that StartByTest method.
**Figure 9**

```java
public ActionResult StartByTest(ActionContext context, ActionTestSimulator simulator)
{
    context.setParameterValue("UserName", "test1");
    context.setParameterValue("Password", "pw1");

    AppspaceInfo appspace = context.getAppspace("assetfolder://MyCRM.appspace");
    RegWinInfo regWin = appspace.GetRegWin("PeopleTab");

    if (mapping.IsAvailable == false)
        throw new Exception("MyCRM must be loaded with People tab selected");

    return MainScript(context, mapping);
}
```

**Start By User Scenario**

This interface is used when you want to allow the user to execute an action directly from Ratchet-X Commander. When implemented in an action, the user can right click that action in Commander (in the Actions tab, see **Figure 18**), and click Execute from the context menu. The action is run and the **StartByUser method** is called to start it. Since there's no existing application context and no RegWins are known to be found on the desktop, the method gets passed only an ActionContext object. This Start By scenario is rarely used, but can be useful when you want an action on-demand and that action is not related to the state of a desktop application. See **Figure 10** to review its relatively simple code definition.

**Figure 10**

```java
#region StartByUser: called when user executes this task from Commander's action tab.

public ActionResult StartByUser(ActionContext context)
{
    return MainScript(context);
}
#endregion
```
Start By CommandLine Scenario (Figure 11)

```csharp
#region StartByCommandLine: started by RatchetX.CommandLine

public ActionResult StartByCommandLine(ActionContext context, string[] CommandLineParameters)
{
    AppspaceInfo appspace = context.GetAppspace(CommandLineParameters[0]);
    RegWinInfo regWin = appspace.GetRegWin(CommandLineParameters[1]);
    return MainScript(context, regWin);
}

#endregion

This interface is quite useful and allows your action to be called directly through a command-line utility. This is typically used in scenarios where another program is launching the action and supplying parameters via a command-line. At runtime, this interface can only be called if the user license file allows it. In some cases, a special license is required called the Aggregation Workstation License, which allows Ratchet-X to be used on a server and acting on behalf of multiple users. Additionally, for this interface to work you need to set the Ratchet-X Commander option CommandLineAccess to "Allow" (see Figure 12).

Figure 12
The command-line utility is launched via **RatchetX.CommandLine.exe** and can be found in the standard install folder. The utility is called using command-line parameters as shown in **Figure 13**.

**Figure 13**
Among the parameters you need to pass, you must supply the action key for the target action, which can be found in Commander by clicking on the action and choosing "Details" (see Figure 14).

**Figure 14**

<table>
<thead>
<tr>
<th>Action Details</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caption:</td>
<td>Address Verification</td>
</tr>
<tr>
<td>Key:</td>
<td><a href="http://ratchetx.com/actions/RatchetX.AddressVerificationAction">http://ratchetx.com/actions/RatchetX.AddressVerificationAction</a></td>
</tr>
<tr>
<td>Modify Date:</td>
<td>9/18/2014 11:14:13 AM</td>
</tr>
<tr>
<td>Location:</td>
<td>installfolder://Samples/RatchetX.AddressVerification.action</td>
</tr>
<tr>
<td>Description:</td>
<td>Verify addresses using QualifiedAddress.com web service.</td>
</tr>
<tr>
<td>Publisher:</td>
<td>RatchetSoft</td>
</tr>
<tr>
<td>Action Window:</td>
<td>Hidden</td>
</tr>
<tr>
<td>Start By:</td>
<td>Mapping: installfolder://samples/ratchetx.addressverification.xmodel</td>
</tr>
<tr>
<td></td>
<td>Action Editor</td>
</tr>
</tbody>
</table>

You can then assemble your command line parameters and call Ratchet-X as in the example depicted in Figure 15. For passing a large or complex set of data, we suggest writing it to a file and passing the file name as argument.

**Figure 15**

```
C:\Program Files (x86)\Ratchet-X Desktop Integration\ratchetx.commandline.exe /command=ActionExecute /actionKey=http://ratchetx.com/actions/RatchetX.AddressVerificationAction /p1=10 Brook Ln /p2=Manhasset /p3=NY
```
Start By Snaggle Scenario (Figure 16)

```csharp
#region StartBySnaggle: started by a snaggle keystroke followed by a screen scrape and pattern match

public ActionResult StartBySnaggle(ActionContext context, SnaggleInfo snaggle)
{
    return MainScript(context, snaggle.Value);
}

#endregion

// RequiredPatterns: Used by Commander to determine if the snagged screen contains data for this action
public string[] RequiredPatterns = { return new string[] { "XLT[0-5](5,6)-A" }; }

This interface allows your action to be started based on the presence of a screen scraped value from an application screen without any defined RegWin or Mapping. The value scraped is be passed into your StartBySnaggle method and conforms to the regular expression pattern you require in the RequiredPatterns property. For more details on Snaggles see Snaggles: Skip the RegWin by finding patterns on any screen.

Start By File Scenario (Figure 17)

```csharp
#region StartByFile: called when user drags and drops a file on this action in Commander

public ActionResult StartByFile(ActionContext context, string[] filePaths)
{
    return MainScript(context, filePaths);
}

#endregion

This interface allows you to enable the user to drag and drop a file onto the action's label with Commander's Action tab (see Figure 18). One or more files can be dropped and, if this interface is implemented in your action, these files will be passed as a string[] to the StartByFile method.
Figure 18

Ratchet-X Commander

- **Address Verification**
  Verify addresses using QualifiedAddress.com web service.

- **Process Files**
  Add invoices from drop

Ready  R:0(0)k:0(0) m:0(0)...
Packaging and redistributing a Ratchet-X solution

Once Ratchet-X has been configured and fully tested on your test machine, it's time to deploy the solution to users. Ratchet-X requires no servers of any kind to support your deployment. All the files and settings are bundled into a custom Windows setup package, which gets installed on each user's desktop. Any revisions you make to your assets get deployed to your users through the same process: a package installation which overwrites the existing installation. All of this work is done by the **Package Editor**. The Package Editor takes your assets, settings, and license key, and inserts them into a standard Windows install file (**MSI**), thus producing a turn-key setup ready for redistribution and installation.

**The Ratchet-X Local Desktop Footprint**

In order to fully understand the deployment process, you should first familiarize yourself with some important asset folders, files and core concepts that make up a Ratchet-X deployment.

**Install Folder**

This is the folder within which the Ratchet-X binaries are installed. This is usually found in `C:\Program Files (x86)\RatchetSoft\Ratchet-X Desktop Integration`, but can be changed during installation.

**Packages Folder**

This folder contains any assets you use during package creation using the Package Editor. It's the destination for the packages you create as well as the package definition files saved by the Editor. This folder is created the first time the Package Editor runs.

**Development Assets Folder**

In Ratchet-X terminology, assets refer to action, appspace and xmodel files. These files are created using the associated Ratchet-X toolkit editors and saved to your local storage. By default, all editors save to a special folder called the **Development Assets** folder. This folder is created automatically and located directly underneath the Windows special folder **Documents**. An example of its path might be `C:\Users\Sally\Documents\Ratchet-X Development Assets`, but it varies based on operating system version. Although it's possible to store your assets in any folder in your local storage, we strongly recommend using this folder to ease distribution and testing. This folder contains the assets with which you work in the various toolkit editors while they're under development only. Once you distribute your solution via the Package Editor, these assets become
read-only and will then be found in the **Installed Assets** folder. The Development Assets folder is created the first time any toolkit editor is loaded, and subsequently, will not appear on most end-user runtime machines.

**Installed Assets Folder**

All runtime assets are found in the **Assets** folder underneath the Install Folder. This folder contains read-only assets (*appspaces/actions/xmodels*), that have been bundled into the installation package by the Package Editor. These files should never be altered directly. To change them, recreate your package and reinstall.

**App User Data Folder**

Every Windows application is assigned a special folder by the Windows operating system which is intended to be used by the application to save local data. This folder is called the **App User Data** folder. The folder is stored in paths that are created by Windows and are very different depending upon the version of Windows you're running. Ratchet-X provides the location of its User App Data Folder in **Commander's Help|About box**. This folder contains the user's configuration file, **RatchetX.Config.xml**, as well as various other files needed for different features.

**RatchetX.Config File**

Stored in the App User Data Folder, this file is created by Ratchet-X Commander to store any settings/preferences you configure while using Commander. Settings include registered action URLs, registered appspace URLs, Tools|Options menu settings, scanner interval and magic button preferences. It's important to note the configuration file contains URL references to the actions and appspaces. It DOES NOT contain the actions and appspaces themselves. If you intend to copy your configuration file to other machines, your URLs must not contain any local references. For more information, see **Supported URL Schemes** below.

**Supported URL Schemes**

Throughout the Ratchet-X platform, URLs are used to identify the location of asset files so it's important to understand their proper usage. The supported URL schemes are:

- http
- file
- installfolder
- assetfolder
http
The scheme http:// is used to reference any web accessible resource and is resolved using Internet Explorer. These assets are downloaded each time Commander starts. The assumption is that you've stored your assets on an internet or intranet web server that can be easily referenced via an http URL.

file
The scheme file:// is used to reference assets which are local or available on a local area network. However, be careful how you use this scheme because local references may not resolve on another user's machine and using them may result in a non-portable configuration. For maximum portability, see scheme assetfolder:// below. Note: local network locations that are accessible via a consistent path such as mapped drives or UNC paths are acceptable and can be referenced like file://S:/Asset/Appspace1.appspace or file://shared1.myserver/Asset/Appspace1.appspace.

installfolder
The scheme installfolder:// refers to the folder, selected by the user during install, into which the package is installed. The default is C:\Program Files (x86)\RatchetSoft\Ratchet-X Desktop Integration. This scheme is used only to reference sample assets found in the subfolder Samples, referenced like installfolder://samples/ratchetx.addressverification.action.

assetfolder
The scheme assetfolder:// is the most important and highly recommended scheme. Unique to Ratchet-X, this scheme is dereferenced at runtime for each user. By using this scheme for your references, you're assured your assets' references are portable to other machines and ready for deployment using the Package Editor as "bundled" assets. There are two physical folder locations that are used with this scheme, Development Asset Folder and Installed Asset Folder.

The Development Asset Folder is location within which assets are stored while in development. Its path starts at the special Windows folder Documents (or My Documents), and is contained in a subfolder called Ratchet-X Development Assets.
The **Installed Asset Folder** is contained in a subfolder named **Assets** under the install folder. This is the runtime home of "bundled" assets when deployed using Package Editor. With two possible physical locations for assetfolder assets, which one does Ratchet-X use? The logic depicted in Figure 1 is used to locate the "assetfolder" asset:

**Figure 1**

![Flowchart](image)

This duality is critical to the functioning of the assetfolder scheme. The same reference (ex: "assetfolder://myappspace.appspace"), references the development folder file during initial
development. But then, after packaging and installation to a new machine, it also references the installed asset folder of the runtime machine. It's possible to have two copies of the asset file, one in the development folder and one in the installed folder. When this happens, the development copy always supersedes the installed copy. This allows you, while debugging on a runtime machine, to make changes to an installed appspace, save it to the development asset folder and test it with confidence you're always testing the development copy. Commander will show the words (Dev) or (Installed) following the asset location URL to avoid confusion (see Figure 2).

**Figure 2**

<table>
<thead>
<tr>
<th>Action Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caption: Address Verification</td>
</tr>
<tr>
<td>Key: <a href="http://ratchetx.com/actions/RatchetX.AddressVerificationAction">http://ratchetx.com/actions/RatchetX.AddressVerificationAction</a></td>
</tr>
<tr>
<td>Modify Date: 4/15/2015 10:26:56 AM</td>
</tr>
<tr>
<td>Location: assetfolder://Test1.action (Dev)</td>
</tr>
<tr>
<td>Description: Verify addresses using QualifiedAddress.com web service.</td>
</tr>
<tr>
<td>Publisher: RatchetSoft</td>
</tr>
<tr>
<td>Action Window: Hidden</td>
</tr>
<tr>
<td>Start By: Mapping: installfolder://samples/ratchetx.addressverification.xmodel</td>
</tr>
<tr>
<td>Action Editor</td>
</tr>
</tbody>
</table>

Once you've tested your changes (and copied the modified asset back to your development environment for repackaging), remember to remove this file from the development folder so it will not supersede the fixed asset you'll be redeploying. Commander does a runtime check for this state and warns the user if superseded assets are detected (see Figure 3).

**Figure 3**

Developer Note: One or more assets is being sourced from the development asset folder. This is normal during development but can cause confusion for runtime installations.

OK
The Package Editor

The Package Editor is started from Commander's Tools menu. Once loaded, it behaves like a wizard - guiding you through the necessary steps and resulting in a build process which creates the installer package. The following steps through the process.

Figure 4

Edit A Package

This wizard guides you through the creation of a custom installation package.

What is an installation package? It's a bundle that includes Ratchet-X software and your configuration files (actions/Apps/Logs/etc...) bundled into a single installation package (MSI) that can be installed on each user's desktop.

To begin, you must either select an existing package definition (that you created earlier) or you can create a new one.

Create New Package  Based on this machine's configuration

Open Existing Package Definition

The first step is to either select an existing package definition file (if you've already created one), or to define a new one. When you select Create New Package, make sure to check the Based on this configuration checkbox which will save you some time by pre-populating some of the next steps with appropriate selections using this machine's configuration. Whether you're creating a new package or editing an existing package, the remainder of the steps are the same. To proceed, click Next
The next step is to select the **Seed Package**. This term refers to the Windows installer file that will be used as a starting point for your integration. Remember, the packaging process starts with a pre-existing Ratchet-X installer into which your configuration files are injected. We call this file the **Seed Package**. Generally, this is the same file you would download from our website for evaluation. It contains no appspaces or actions, and has a default (empty) configuration. The file name is **RatchetX.FullSetup.msi** and you can stay up-to-date by selecting the latest production releases from the radio buttons and clicking the download button to obtain it. Alternately, you can select a specific version (if you're risk averse), and always download that version by selecting the second radio button. Another choice would be to work with a local copy of the file by downloading it once from the website, storing it locally and referencing it in the textbox provided. Once you have selected the seed file, click **Next** to continue.
Now you need to select the source of the **RatchetX.Config.xml** file (defined above), that will be bundled into the package. By choosing the option **Use Current Config File** on the first screen, the Package Editor will select the file stored in the current App User Data folder which contains the exact configuration file of the running instance of Commander. This is often the easiest way to go if you want to simply clone this test environment for distribution.

An alternative method would be to copy your configuration file to a web or file server and not bundle it into the package. To do this, you must provide the config file's URL location which will be written to the settings and instructs Commander every time it loads to download the configuration file. This scenario adds complexity and a single-point-of-failure but has the benefit of allowing you to change the user's configuration without redistributing the package. Merely updating the configuration file on the server will make it live to your users next time they load Commander. For a full explanation of this scenario, see [Hosted Asset Deployment](#) below.

Yet another option is to provide a specific local path to your configuration file. You start this process by going to Commander's Tools menu, selecting **Export Configuration**, and then selecting an appropriate folder (Ratchet-X Packaging folder is best). That configuration file is then frozen in time. The benefit here is you can setup multiple configuration scenarios (for different user groups), and create a different package, with a different configuration file, for each user group.

After you've defined your configuration file, click **Next** to continue.
In the next step, you can optionally provide Ratchet-X license key information so your users are not prompted for it at runtime. This license information is provided to you at the time of purchase. Be careful to copy it exactly as presented to you since the Package Editor will not validate the key. Validation happens only at runtime on the user’s computer.

Click **Next** to continue.

**Figure 8**

Here you can optionally add any appspace, actions or xmodels which will be distributed with this package. These files will be installed in the ‘Assets’ folder under the INSTALLDIR.
In this step, you must provide the asset files you want to bundle into this package. Asset files include the actions, appspaces and (don't forget), xmodels. If in step one you chose the, **Based on this machine** option, all assets from the AssetFolder will be pre-populated. Note: if you choose to distribute no assets (None), it is assumed your assets are hosted somewhere and your configuration file contains accurate URL references (see **Hosted Asset Deployment** below).

Click **Next** to continue.

---

**Figure 9**

![Add 1 Support Files](image)

Here you can optionally add any necessary files like Active-X, dlls, .NET assemblies or executables that are used by your actions or connectors. These files will be stored in the "INSTALLDIR" folder.

---

In this step, you can optionally add any additional files you may want distributed with your package. During installation, all of these files are copied to the Install Folder except any OCR training files (extension **traineddata**), which will be copied to the **tessdata subfolder**.

Click **Next** to continue.
Now you're ready to build your package. By clicking the **Build/Save** button, the wizard begins gathering all the bundled assets and inserting them into the seed installer file. When completed (see **Figure 11**), the wizard outputs the new package to the package folder which can be loaded by clicking the link **Open Package Folder**.

The output folder contains two new files, your package file itself (extension **MSI**), and the package configuration file (extension **RXPACKAGE**). The base file name of each will be the same value.
supplied as the build name. It's important to save this package configuration file so it can be used as the basis for any future updates you make. This file contains the **Windows Installer Product Code** needed to perform an update of the original package. If you lose this file, you can always recreate it, but the user will be required to install his existing package prior to installing the new one.

**Hosted-Asset Deployment**

Although RatchetSoft recommends bundling your assets and config file directly into the package, some choose a different approach. In environments where asset updates (action or appspace), are expected with frequency, it’s often preferable to avoid the "installation" approach to asset deployment and host your assets and configuration file on a file or web server in your local area network. Note this approach does create a single-point-of-failure.

If you choose to use this option, simply copy your assets and config file to the server, create appropriate URLs and use those locations when you:

- register actions in Commander
- register appspaces in Commander
- add mappings (to xmodels) in the Appspace Editor

Once you have your test machine working using these (http or file) URLs you need to export a copy of your configuration file to that server as well. At this point, you've gathered your appspaces, actions, xmodels and config file on that server and you're ready to create a hosted-asset package using the Package Editor. The step-by-step explanation above is the same with the following two exceptions. First, when prompted for a Configuration File, provide the URL to your RatchetX.Config.xml file on the server. Second, when prompted for Assets, choose "None".

The package that gets created will not contain any assets and your deployment will have a dependency on this file server, but your end-users will have a fresh copy of your config file and assets downloaded each time they load Commander. This will provide the benefit of allowing you to rollout changes to these files without going through the MSI installation process.
Commander Status Bar: What do those characters at the bottom of Commander mean?

Commander's **status bar** provides you with some basic diagnostic information.

When Commander is active and scanning the desktop for application screens, the word "**Ready**" appears on the left side of the status bar. In addition, scanning diagnostic information is presented on the right side of the status bar. The legend below describes the information:

- **R** - Represents the number of regwins (short for "registered windows" and usually correlates directly to the number of application screens), for which Commander is scanning. The number displayed in parenthesis next to the value after "R" represents the number of regwins.
Commander has currently found. For example, R:3(1) means Commander is scanning for three regwins, one of which has been found.

- **K** - Represents the number of keyboard-based shortcuts available across all of the regwins Commander has found during its last scan. The value in parenthesis denotes the subset of keyboard shortcuts that are currently active. For example, K:4(2) means the total number of shortcuts spread across all the regwins Commander has found during its last scan is four, two of which are currently active. The other two shortcuts mostly likely require the cursor be placed in a specific field in order for those shortcuts to become active.

- **M** - Represents the number of mouse click-based shortcuts available across all of the regwins Commander has found during its last scan. The value in parenthesis denotes the subset of mouse click-based shortcuts that are currently active.

The dots rendered to the right of the last part of the M value lets you know whether or not Commander is scanning the desktop for regwins. If the dots are cycling, Commander is scanning. If the dots are not cycling, Commander is not scanning.
Snaggles: Skip the Regwin by finding patterns on any screen

On occasion, you may find it handy to relate a feature (action), to a piece of information on any application screen rather than relating the feature to information on specific application screens. For example, let’s say you want an action to process a phone number regardless of which application screen it appears upon. Using regwins, markers and snippets, you would need to create a reqwin for every screen upon which a phone number might appear. Based on the number of applications and screens a user works with on a daily basis, this becomes impractical. To accommodate this requirement, Ratchet-X has a facility called Snaggles. Snaggles allows you to associate an action with the presence of a formatted string (based on a regex), regardless of which screen it appears on thus eliminating the need to create a regwin for each screen.

Here’s how it works. When you navigate to an application screen and press the snaggle key, (assuming a snaggle key has been defined in Commander’s Tools | Options | SnaggleKey property), the Snagging... screen overlay is placed on top of the application screen and scans the screen for any string of data that matches the regex pattern defined in any action that invokes the Start By Snaggle interface (see Figure 1).
If no matches are found, Commander does nothing and closes the overlay window. However, if one or more matches are found, Commander highlights all the data strings that match the pattern and pauses giving you an opportunity to execute any one of the snaggles found. To see which action will be executed for a given found snaggle, position your mouse cursor over the highlighted data. When you do, you’re presented with the snaggle execute screen (see Figure 2).

Figure 2
To execute the selected snaggle, click the mouse.

Note that if you have Commander’s **Tools | Options | SnaggleOverrideMode** option set to **Auto-Override**, and there is only one snaggle found on the current application screen, the associated action will be automatically executed bypassing the snaggle execute screen.
Sounds: Please tell me how to turn them off

Ratchet-X issues various audio cues to let the user know that Ratchet-X is doing something. These cues include:

- The appearance of the magic button in an integrated application.
- Pasting of data into the screen fields of an integrated application.
- The execution of a snaggle.

While these audio cues can be helpful, some may find them annoying. You can turn off the audio cues by modifying Commander’s Sounds configuration settings.
My Configuration File: What is it? Where is it? How do I save and share it?

All of your Commander options settings and references to registered assets are stored locally in a Commander configuration file. This configuration file is named **RatchetX.Config.xml** and is stored in your Windows managed User App Data Folder. In order to find the precise location of this file, in Commander, click **Help|About...** The location of this folder is displayed on the line that starts with the text, **User App Data Folder**. While you should never have to directly modify the contents of this file, it is important to be aware of this file, especially if you are developing Ratchet-X solutions and creating install packages.

When creating a Ratchet-X integration, there inevitably comes the point when you want to package up your work and distribute it to other users. The most important part of the packaging process is determining which assets to distribute and how to configure certain options for specific user groups. The best way to do this is to configure Commander on your workstation the way you want it for the given user group (including all desired Commander option settings and required assets), and then export your configuration file. The exported configuration file will then be included in the package you create for the targeted user group. For more on creating Ratchet-X packages, [click here](#).

To export your configuration file, in Commander, click **Tools|Export Configuration...**. From there, you simply save your file for use later on when you create your package.

Finally, Ratchet-X also gives you the ability manually import a configuration file. To do so, click **Tools|Import Configuration...**. When the file is imported, assuming the assets referenced in the file are accessible using the same paths as those defined on the workstation from which the configuration file was exported, your version of Commander should be configured to match that of the imported configuration file.
ClickIts: When an action needs to visit many screens to do its thing

As automation scenarios grow more complex, they inevitably require the ability to navigate from screen-to-screen by clicking buttons or pressing keys to navigate from one screen to the next. We describe this as **navigation** and it has, at its core mechanism, the **ClickIt**. ClickIts are defined in the **Appspace Editor** and are a part of a RegWin. You typically define and test ClickIts in the Appspace Editor, then reference them in the Action Editor from code. ClickIts are defined by selecting the **ClickIts** tab and selecting either **New** or **Edit** from the context menu depending upon what you want to do (see **Figure 1**).

**Figure 1**

![ClickIt Editor dialog](image)

**Figure 2** below depicts the ClickIts Editor dialog.
The process is much like defining a snippet or marker because it starts with selecting a **ScreenLink**. Whether you drag and drop or use the **Advanced Path Editor** and define a custom path, you're selecting the screen object which will be the target of your click. Typically this is a button or a link, but it can be anything. After selecting the Screenlink and naming your ClickIt, you get to define the **Click Style**. There are three click styles:

- AccessibleAction
- WinAPI
- Macro
**Accessible Action Style**

Accessible Action is the preferred and most commonly used click style. It uses the Windows Accessibility API to perform the node's default action. The main benefit of using this style is that, when it works, it can work on buttons which are scrolled off-screen. The main risk is that some applications don't return from this call immediately and can block access until some navigation completes.

**WinAPI Style**

WinAPI style uses the Windows API to send a Windows WM_LBUTTONDOWN and WM_LBUTTONUP message to simulate a click. When supported by the application, WinAPI style is very reliable and shows no UI artifacts (visible mouse or other). However, it's dependent on a target application's message handling the call which is not always an option.

**Macro Style**

When the first two styles do not work, macro style is your best hope. That's because this method physically moves the cursor to the click location and drops a click at hardware level. While this should always work, the disadvantage of using this style is that a visible cursor is always displayed which makes for a slightly less elegant UI experience. Macro Style provides a default value for the macro syntax which moves the mouse to the upper left corner of the ScreenLink and clicks the right mouse button. Using the Edit button, though, you can create any macro you wish. Since macros can include multiple keystrokes and different types of mouse clicks (right, left, wheel), you can do quite a bit using this click style. In fact, a typical macro might target the Root ScreenLink of a window and the macro may send Alt+F then X. This would close a standard Windows program. In this case, the ClickIt contains no clicks at all and is merely a series of keystrokes which perform some navigation task (see Figure 3). So keep in mind, though named "ClickIts", the functions they sometimes perform may not actually involve any mouse clicks. For more information on macro syntax see Macros: mouse clicks and keystrokes that go with the flow.
When you drop down the **Click Style** dropdown shown in figure 2 above, you won't see anything called **CustomClick**. This is because, until you define a custom click, no entry is shown in the Click Style dropdown list. But you can create one using code by overriding a special method in a **Connector** called CustomClick (see **Figure 4**). Once you've got your custom click connector compiling, it will then appear in the ClickIt Editor's Click Style dropdown and you can assign it to do your complex navigation.
Once you've got your ClickIt defined, tested and saved, it can be referenced in either your action code (in Action Editor), or your connector code (Appspace Editor). Actions use ClickIts in a straightforward manner. See Figure 5 for a few usage examples of calling a ClickIt.
Connectors can also reference ClickIts. In Figure 6 you see an example of a Snippet’s Extract method which needs to first click a tab, read additional data, and then click another tab to return.
In Figure 7 you can see an example of a ClickIt's CustomClick method which calls one ClickIt or another based on evaluating a condition.
```csharp
using System;
using System.Windows.Forms;
using System.Drawing;
using RatchetX.Scripting.Connectors;

namespace MyCRM
{
    public class MyCustomClicker : ConnectorRuntime
    {
        public override Point CustomClick(Screenlink screenlink, bool doubleClick)
        {
            bool someCondition = true;
            if (someCondition)
            base.Click("EnterButton"); // referencing another ClickIt (beware recursion)
            else
            base.Click("ExitButton"); // referencing another ClickIt (beware recursion)
            return Point.Empty; // can return actual point of click if desired
        }
    }
}
```
Finding Bitmaps: When all you have is an image

It's easy to overlook this simple feature, but hard to deny its importance when needed. The ScreenLink object, available inside connectors, has a method called `FindBitmap` whose signature is shown below:

```csharp
public Rectangle[] FindBitmap(Bitmap targetBitmap, Rectangle searchZone,
    float similarityThreshold, bool bringToFront,
    TimeSpan timeout)
```

This simple method can be a life-saver when working with an uncooperative target application. Imagine you have a screen like the one shown below in **Figure 1**, and you need to click the "phone" button.

**Figure 1**

![Screen capture showing the "phone" button highlighted](image-url)
The challenge with this case is the application is not well-behaved and no ScreenLink path can be created. Making matters worse, the button bar can be multiple lines and the phone button may not be in the same location, so using a ScreenLink path with an AltRect option will not work because it assumes a fixed location. So how do we consistently find this button?

The `ScreenLink.FindBitmap` method allows you to pass in a bitmap image and conduct an exhaustive pattern matching search within the boundaries of that ScreenLink. Internally, Ratchet-X is taking a screen shot of the ScreenLink zone and performing an image search for the supplied bitmap. The match does not need to be exact or else slight variations from machine to machine in color depth or DPI could break it. Instead, you supply a similarity threshold defining the level of variation you'll accept. The similarity threshold must be between 0F and 1.0F with a value of 1.0F means it must be an exact match. Typically 0.8F is a good choice but trial and error needs to be used to know for sure. The method returns an array of rectangles which are the coordinates of any matches. Once you have the screen coordinates, sending a mouse click can be done using a `ClickIt`.

If you choose to use this method, you must consider a few things. First, an exhaustive image search may be slow. Search performance depends upon two things - the size of the ScreenLink.Zone to be searched and the size of the image to find. As a rule of thumb, you should make the search zone as small as possible. If you're lucky, and the outer toolbar object itself has accessibility, you can confine your search to that ScreenLink. If not, and your only ScreenLink defines a much larger area, you may need to do some math and calculate a rectangle which will define the search zone within your ScreenLink zone that will limit the search to a manageable portion of the image. For example, just the topmost third of the image. This will improve the performance of the search and is essential to making this technique workable.

Let's look at some examples of how this method may be called. The first example below assumes we have a ScreenLink defined for the toolbar itself which represents an appropriate zone (fast to search).
Let’s assume in this example we do not have an appropriate ScreenLink and have to use the Root ScreenLink. With such a large zone, the search time will be many seconds so we first calculate a smaller zone within the search area. This smaller search zone is defined as an offset rectangle. Its coordinates are not desktop coordinates but relative the ScreenLink rectangle itself so a rectangle of (X=0,Y=0,Width=100 and Height=100), starts at the upper left of the ScreenLink rectangle and extends 100 pixels in x and y.
Another consideration when using the ScreenLink.FindBitmap method is deployment of the bitmap file. Once you have this working, you need to get that bitmap file distributed with your appspace. Currently, the only way to do this is to include it as Additional Files when you create your redistributable package using the Package Editor. By adding the bitmap file to the package, it is copied to the install folder of each installation and available using the same method shown in this example.

What about ScreenLink.FindBitmap and its potential use in custom markers? There are scenarios where this could be useful. Imagine an onscreen graphic or button being shown depending on some condition, and we want our marker to be found only if that button is showing. If that button or graphic is unavailable via a ScreenLink path you'll have no alternative. But approach with caution. Image search is described as "exhaustive" because it perform lots of math and could be processor intensive when used with large bitmap or a large search area. If used as a marker it could potentially add a multi-second processing job, every time Commander scans the desktop (which by default is every 1.5 seconds), and result in a drag in system performance. We advise you to use this technique only for markers in RegWins which never show the magic button and therefore are not in Commander's main scanning loop. These RegWins are typically used just for navigation during an action's flow.
Figure 3

```csharp
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
using RatchetK.Scripting.Connectors;

namespace MQAEexpress
{
    public class ButtonSharingEvaluator : ConnectorRuntime
    {
        public static Image PhoneBitmap;

        public override bool Evaluate(ScreenLink screenLink)
        {
            // If not already resolved, find bitmap and store in static variable
            if (PhoneBitmap == null)
                PhoneBitmap = Bitmap.FromFile(Path.Combine(Application.StartupPath, "phone.bmp");

            // Define an offset rectangle which will restrict the search zone to the top 1/3 of the ScreenLink [whole window]
            Rectangle offsetRectangle = new Rectangle();
            offsetRectangle.X = 0;
            offsetRectangle.Y = 0;
            offsetRectangle.Width = screenLink.Location.Width / 3;
            offsetRectangle.Height = screenLink.Location.Height / 3;

            // Perform exhaustive search for matching image.
            // 0.8F means we are looking for 80% confidence or higher
            Rectangle[] zones = screenLink.FindBitmap((Bitmap)PhoneBitmap, offsetRectangle, 0.8f, true, TimeSpan.FromSeconds(10));

            // If a match exists, return true, otherwise false.
            return (zones.Length > 0);  
        }
    }
}
Action Configuration: How to get end user data like passwords and server names

Often, your action may require some configuration data that is unknown at action design-time and therefore cannot be hard-coded in your action. There are a few different ways of accomplishing this, each with its own benefits.

1. Appspace Author Supplied Values - when the data being supplied is not unique to each user of the action, but instead its unique to each integration, the best option is to add appropriate elements to your XModel allowing the Appspace author to supply this data while mapping your XModel to their Appspace. This assumes that your action implements IStartByMapping and has a defined XModel. An example of this would be a Server name. Let’s say you know that the users of your action will all share a common server at each installation/integration. So it makes sense to have the Appspace author set the configuration by adding it as a constant XModel element during the appspace / mapping process. Then, your action simply retrieves the configuration elements as below:

   context.Getsdfsdfasdfsdf

2. User Supplied Values - when the data being supplied can be unique for each user, you can use Action Parameters. Action Parameters are defined by the Action author and can be user edited using Commander.

   How do I create the Action parameters? Simple, just add the IHasParameters interface to your class definition and implement its GetDefaultParameters() method. Here is a brief example

   public class TestAction: ActionRuntime, IStartByTest, IStartByRegWin, IHasParameters
   {
       ...
       ...
       public ActionParameter[] GetDefaultParameters()
       {
   }
There are a number of useful ActionParameter types which handle common types of input like passwords, lists, etc. Here is a list of each type and its usage.
Aggregation Workstation: What is it? Why may I need it?

Ratchet-X is designed to work on a local user’s computer and assumes direct access to all required applications. However, there are times when a user needs access to data, or the ability to initiate a transaction, from a device that does not have direct access to those applications nor is running Ratchet-X. This is where the Ratchet-X Aggregation Workstation comes in to play. Aggregation Workstation allows you to run a Ratchet-X automation on a remote workstation thus freeing the user up from having to have direct access to the integrated applications or the Ratchet-X runtime.

Well that sounds pretty interesting but why would I need such a thing? The best way to answer this question is to look at a sample use case.

**Order Inquiry Checking**

Let’s say you want to make order inquiry capability available to sales reps and customers via the web, but your order inquiry system cannot be web-enabled. Using Aggregation Workstation, you can create a simple web accessible front end interface that merely accepts the inquiry parameters from the user (web page, web services, native mobile device application, etc). This front end interface then in turn makes a web service call to RatchetSoft Clod Relay queue (hosted by RatchetSoft), that is connected to one or more instances of Ratchet-X. When a request is placed in the queue, one of the listening instances of Ratchet-X picks up the request and executes the query via an action. Keep mind, the instance of Ratchet-X that executes the action must have direct access to the order inquiry system. Once the results come back from the action, Ratchet-X places them in the same queue which in turn is picked up by the front end interface and presents them to the user.

The bottom line is many organizations utilize systems that have no programmatic interface which would enable them to make the data contained within available to users whom do not have direct access to those systems. Aggregation Workstation working in tandem with RatchetSoft’s Cloud Relay queuing service allows you to make information bottled up application silos and make them available to any user, regardless of their device he's using at the time he needs the information.
64bit support demystified

Most Ratchet-X version 5.5 features support 64 bit applications. The following describes the features that are not supported and how to work around them.

**How Do I Know My Application Is Running 64 bit?**

If you are unsure if your application is a 64 bit application, you can easily check by performing the following while the application is loaded.

- Run task manager (either by pressing Ctrl-Alt-Del) or by entering Taskmgr into application search box.
- When task manager loads, look for your application in the Applications tab. When found, right click on the task entry and select Go To Process.
- When you do, task manager will switch to the Process tab and your running application process will be highlighted. If the Image Name is followed by *32, that means your application is running in 32 bit mode. If not, then your application is running in 64 bit mode.

The other way to determine if your application is 64 bit is to try to create an appspace for the application screen using the Appspace Wizard. When you select the target window, if the application is a 64 bit application, the message "64 Bit" is superimposed on top of the screen image. This is not an error, but rather a reminder message letting you know your application is 64 bit and that you should consider the issues below.

Now that you have determined for sure your application is 64 bit, here are the things you need to know.

**Magic Button**

The magic button can be rendered in all the same places that is can for 32 bit applications. The only difference you will notice is when you move a window within which the magic button is rendered, the button will not render during the actual move process. However, the button will re-render once you stop moving the window. Other than that, everything works the same.

**Shortcuts**

Since the current version cannot hook the 64 bit window at the operating system level, shortcuts are not supported. This will be fixed in version 6.
**TextScrape**

While the MSAA and OCR methods do support extractions from 64 bit applications, our Scrape connector does not. This will be fixed in version 6.

**Running 64 bit Application in 32 bit Mode**

Any 64 bit application that is compiled to support "any CPU mode", can be stepped down to run in 32 bit mode. Since most business applications do not take advantage of the 64 bit processor, this can usually be done without any noticeable impact to the user. To run a 64 bit application in 32 bit mode, you need to create a 32 bit loader using the Ratchet-X **AnyCPU Loader Generation Utility**. You can load this utility by running RatchetX.AnyCPULoader.exe in the Ratchet-X install folder.

**Figure 1**

![AnyCPU Loader Generation Utility](image)

This utility is used to create a bootstrap loader that runs a .NET application in 32 bits. This assumes that the application is compiled targeting "Any CPU". Just select the executable file and click Generate Loader EXE.

Once loaded, click the **Select** link to select the executable file that loads your application. Once selected, click the **Generate Loader EXE** button. When you do, the AnyCPU loader executable for your application will be created and stored in a folder on your desktop. As the message indicates, you must copy the loader executable into the same folder as your actual application executable. You must do this in order for the loader to find the actual application executable file when run.
RegWin Debugger: What is it? How do I use it?

Ratchet-X comes equipped with an internal debugging action that helps you test and debug your regwins. Think of this debugger action as a "universal action" that once registered, can instantly interact with any appspace without the need to create action specific mappings. The debugger action allows you to test and debug any regwin including all subordinate markers, snippets, clickits, shortcuts and mappings.

Once added to Commander, the debugger action allows you to simulate extracts and pastes, execute clickits, shortcuts and test the integrity of your mappings.

To register the debugger action into Commander, right click in Commander's Action tab and select Add Sample... | RatchetX.RegWinDebugger. Once you've added the debugger action, it's ready to be used to debug any appspace without any modifications required. The debugger action makes its presence known at runtime when you click the magic button from any integrated application. See Figure 1 below for an example of how the regwin debugger alert is generated when the MyCRM sample appspace is registered with Commander, the MyCRM sample application is loaded and the magic button is clicked.

Figure 1
Note that if there are multiple regwins available on the same application screen, the alert window will present a debugger action alert for each regwin.

In order to execute the debugger action, click the alert. When you do, the debugger action dialog displays. See Figure 2 below for an example.

**Figure 2**

![RegWin Debugger: PeopleTab](image)

The debugger dialog places the name of the regwin being debugged in the title bar and presents the following five tabs:

- **RegWin**
- **Markers**
- **Snippets**
- **ClickIts**
- **Mappings**

**RegWin**

The regwin tab contains a panel of read-only information that describes the regwin (as described in the Window tab of the regwin in appspace editor).

**Markers**

The markers tab presents list of the markers used in the regwin. As you click through the list of markers on the left, the panel to the right displays the marker's validation condition (if one exists), and whether or not the marker is mandatory. To highlight the marker in the integrated application, click...
Snippets
The snippets tab presents a list of the snippets used in the regwin. As you click through the list of snippets on the left, the panel on the right allows you to perform the following three functions: Highlight, Extract data from or paste data to the selected snippet. To highlight the selected snippet, click the Highlight link. To extract data from the selected snippet, click the Extract link. To paste data to the selected snippet, type the data you want to paste to the snippet and click the Paste link.

ClickIts
The ClickIts tab presents a list of the clickits used in the regwin. In order to test a clickit, select the clickit you want to test from the list on the left and then click the behavior you want to test by clicking the associated link on the right. Clicking Highlight will highlight the clickit. Clicking the Click link will click the clickit (wow, that’s a lot of clicks). Clicking the Double Click link will double click the clickit (a lot here too).

Mappings
The Mappings tab presents a list of the mappings used in the regwin. In order to see which snippets are mapped to a given mapping, select the mapping item from the list on the left and then click the link on the right you want to test. To highlight the mapping (placing a single rectangle around all the snippets contained in the mapping), click the Highlight link. Note that it’s possible that not all the screen nodes contained within the highlighted rectangle are contained in the actual mapping.

To extract data for all the snippets contained in the mapping, click the Extract link. When you do, you’re presented with a listing of all the mapped snippets along with their current values in the integrated application. See Figure 3 for an example of Extract link click.
Figure 3

To close the debugger dialog, click the **Close** link.
What is MSAA and why should I care?

When people first discover Ratchet-X, many often associate what it does with an application integration technique known as “screen scraping”. Screen scraping is an integration technique in which a computer program extracts data from the display output of another program. While originally developed as a passive mechanism to get information off of mainframe green screens for use in other applications, the use of screen scraping is alive and well today for getting data out of client server and browser-based applications. The biggest negative associated with screen scraping routines is they rely on unstructured and variable user interfaces so they tend to be brittle and break often. This is precisely the reason Ratchet-X does not use traditional screen scraping as its primary method to exchange data with application screens (though the analogy is appropriate for gaining a high-level understanding of what Ratchet-X does).

So how is Ratchet-X different from screen scraping and why is it better and more reliable? Glad you asked. While screen scraping strictly adheres to screen positioning at the pixel level, Ratchet-X utilizes a little known Windows API called Microsoft Active Accessibility (MSAA). MSAA has evolved with the release of the Microsoft .NET framework version 3.0 and is now called Microsoft User Interface Automation (MUIA). For the purpose of this discussion the difference to you is irrelevant so we will refer to the MSAA API for the balance of this section.

MSAA is typically used by applications such as screen readers to help visually impaired computer users interact with software. Technically, MSAA breaks down an application screen into specific elements (called nodes), that can be addressed through an alternative interface such as speech. In other words, the user can verbally instruct an application to click the File menu and open a file or select New York from a combo box.

Since this API is widely supported (meaning most applications that run under Windows inherit this capability), RatchetSoft has elected to piggy-back on this technology in order to "understand" an application's UI elements. Once Ratchet-X understands what those elements are, it can then manipulate them very much the same way screen readers do. The only difference is that screen readers manipulate UI elements (known as MSAA nodes), through speech while Ratchet-X allows the nodes to be manipulated by other applications (through Ratchet-X and the interaction with actions).

Within the context of Ratchet-X, each MSAA node possesses the properties listed below.
### Table of Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Object name defined for the node within the application. For example, a field that accepts a phone number might possess the MSAA name, &quot;phone&quot;. This name is defined by the person who developed the application you're using. Keep in mind that developers are not obligated to supply node names so don't be surprised if a node does not possess a name.</td>
</tr>
<tr>
<td><strong>Role</strong></td>
<td>The type of UI element. For example, static text for label, combo box, editable text for field, etc.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>Refers to the value associated with a node. Within Ratchet-X, it usually contains the value currently stored in a screen field. For web pages, quite often, the value is actually reflected in the Name property rather than the value property. The more appspaces you build, the more you'll see patterns emerge.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Description of node as supplied by the developer of the application. This is often blank.</td>
</tr>
<tr>
<td><strong>Class</strong></td>
<td>The Microsoft Windows class value of the node.</td>
</tr>
<tr>
<td><strong>Visible</strong></td>
<td>Flag denoting whether or not the node is currently visible on screen.</td>
</tr>
<tr>
<td><strong>Available</strong></td>
<td>Flag denoting whether or not the node is available for access. Sometimes a node may be visible on a screen but is not available for manipulation.</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>Denotes associated function that will be performed if the node is &quot;executed&quot;. For example, if the node is a button, chances are, its default action will be a button press.</td>
</tr>
<tr>
<td><strong>Selected</strong></td>
<td>Flag that denotes whether or not the node is currently selected. Mostly used within Ratchet-X to identify if a specific screen tab is selected.</td>
</tr>
<tr>
<td><strong>Read Only</strong></td>
<td>Flag that denotes whether or not a node is read only.</td>
</tr>
<tr>
<td><strong>Checked</strong></td>
<td>Flag denotes whether or not a check box node is checked. Applies only to check boxes.</td>
</tr>
</tbody>
</table>

Some of these properties can be used as the basis for performing validations on MSAA nodes. There are a few other properties available and will be addressed in this guide where applicable.

To summarize, Ratchet-X looks at screens not as unstructured collections of screen elements dictated solely by the way they appear on the screen, but rather a structured, hierarchical tree of property governed nodes that have a relationship to the other windows and nodes on the screen. While MSAA is far
from perfect, it's pretty reliable and almost guaranteed to be available for most applications developed since 2000 running over Windows 2000 and better.

For more information on MSAA and MUIA, visit these links:

Note in Ratchet-X, you will see terms like MSAA node, screenlink and UI elements. In most cases, these terms can be used interchangeably but are referred to in different terms in order to make the description more relevant to the intended audience of the documentation.
Screen Simulator: Seeing is believing

The Ratchet-X Screen Simulator (simulator), allows you to create a demonstration of a prospective Ratchet-X integration project without having direct access to the runtime environment or target application(s). The simulator allows you to take a screen shot of a given application screen and "Ratchet-X enable" it for the purpose of conveying how that application will look and behave when participating in a Ratchet-X integration. Currently, Ratchet-X screen simulations support basic data extracts and pastes.

Loading The Screen Simulator

In order to load the simulator, you must run the application `RatchetX.Screen.Simulator.exe` which resides in the Ratchet-X install folder (typically located in "c:\Program Files\Ratchet-X Desktop Integration" for 32bit Windows operating systems or, "c:\Program Files (x86)\Ratchet-X Desktop Integration\" for 64bit Windows operating systems).

When the simulator loads, you’re presented with following interface depicted in Figure 1 below.

Figure 1

The first thing you'll notice is the simulator is comprised of two linked windows. The window on the left is the main application window from which you initiate all application activities and configure object properties. The linked window on the right is called the "design surface". The design surface is the
window within which your simulated screen will appear, be manipulated and ultimately run in simulation mode.

When the simulator loads, it's automatically placed in **New** mode where you can create a new simulation. The first thing you need to do when creating a new simulation is supply an application screen image. To do so, click the **Screen Image** property within the **Screen Capture** property grouping. When you do, an ellipsis appears to the right of the property field. Click the ellipsis to proceed. When you do, the **Capture Screen** window appears. From this window, you can capture the screen image in one of the following two ways:

**If You Have The Application Running On Your Desktop...**

If the application is running on your desktop, you can capture the screen by clicking the crosshairs icon and drawing a rectangle around the screen region you want to capture. In most cases, you want to grab the screen region starting in the upper left corner of the screen just below the titlebar and slightly to the right of the window's left, outer-most border (see **Figure 2** below). Note the Capture Screen displays for you a magnified version of the crosshairs' location so you can fine tune your grab.

**Figure 2**
Once you've clicked on the region's starting location, drag a rectangle around the entire region you want to capture. In most cases, you want the region's end location to be the lower most right corner of the application screen just above and to the left of the window's outer-most border. **Figure 3** demonstrates this concept.

**Figure 3**

Once you've drawn a rectangle around the region you want to capture, release the mouse button to complete the capture. When you do, the captured region is rendered in the design surface. See **Figure 4** below.
If You Do Not Have The Application Running On Your Desktop But Do Have Access To A Screen Image...

If you have access to a screen image only, click the **Open Image File** link in the Capture Screen window. When you do, you're prompted to open the image file and perform the same steps described above to capture the actual screen region. The simulator supports most commonly used graphic file formats.

**Hiding The Capture Screen Window**

If the screen region you're capturing takes up most of, or the entire screen, you may find it necessary to hide the Capture Screen window so it doesn't appear in your screen simulation. To do this, check the **Hide during capturing** check box before you attempt to capture the screen region. When you do, the Capture Screen dialog will be hidden during the screen capture process.
Finalizing The Simulation Screen Appearance

Once you've captured the screen you want to use for your simulation, you need to supply the simulator with a few more pieces of information before you can start activating data fields. Remember, in most cases, you merely captured the client area of the application screen you want to simulate thus leaving out the titlebar, application icon, titlebar text and control buttons. In order to simulate your application as accurately as possible, you'll need to provide the simulator with these pieces of information as well. You do this by configuring the properties described below.

Properties

TitlebarIcon

You add the application icon to your simulation by clicking the property data field followed by clicking the ellipsis. When you do, once again, you're presented with the Capture Screen window. From here, draw a rectangle around just the icon portion of your running application screen or screen image just as you did with the original screen image capture. Make sure to capture just the icon portion of the image. **Figure 5** below depicts capturing the icon.

**Figure 5**
**ControlBox**
The ControlBox property allows you to enable or disable the appearance of the titlebar icon, standard windows control context menu associated with the application icon, and the window sizing controls in the titlebar.

**MaximizeButton**
The MaximizeButton property allows you to enable or disable the rendering of the titlebar maximize button. Note that while the simulator allows you to render a maximize button, it will not work when the simulation is running since the simulation always conforms its runtime size to the size of the application screen. This button is for display purposes only.

**MinimizeButton**
The MinimizeButton property allows you to enable or disable the titlebar minimize button. The minimize button does work when the simulation is running.

**TitlebarText**
The TitlebarText property allows you to enter a string that simulates the text that appears in the application’s titlebar.

Once the screen simulation appearance is completed, you’re now ready to activate data fields.

**Activating Data Fields**
The purpose of creating a Ratchet-X screen simulation is to demonstrate to potential users how their application will look and behave once it’s Ratchet-X enabled. To achieve the full effect, you need to activate data fields so you can simulate data extractions and pastes. To do this, you must activate data fields that actually "overlay" the defined screen image. Once these fields are activated, you can then build an appspace for the simulated screen and map it to an action.

To add a field, click the add button on the simulator button bar 📝, or click Field | Add Field. When you do, an overlay field is dropped on the design surface over your screen image with a default configuration. See Figure 6 below.
Next you need to position, size and configure your field.

**Positioning And Sizing Fields**

To position a field, click on the field to select it (as denoted by the rectangle handles), and drag it over the field contained in the underlying screen image you want to simulate. You'll know you're in position mode when the cursor changes into a cross arrow. Using the image depicted in Figure 1 as an example, if you
want to include the Add Line 1 field in your simulation, drag the red rectangle over the field portion of the Add Line 1. In this case, that is the white portion of the field that contains **1049 Park Avenue**. See Figure 2 below.

**Figure 7**

Once you've positioned the field properly, you now need to resize the field so it matches the existing field's size less one pixel all around so your field does not encroach upon the field's border. To resize the
overlaid field, grab the handle that coincides with the position on the field from which you want to initiate the resize and drag the handle to the desired position. You'll know you're in resize mode when the cursor turns into a diagonal arrow. Again, using Figure 2 as an example, you want to resize the red rectangle so its right border is one pixel to the left of the field border. See Figure 8 below.

**Figure 8**
Finally, you may need to adjust the height of the field so that it fits within the upper and lower border of the field. To do this, grab either handle in the upper or lower middle portion of the red rectangle and drag it either up or down depending upon what you need. Again, make sure the red rectangle is positioned within one pixel of the field border.

Note that while the example above demonstrates how to manually position and resize an overlaid field, you can also perform the same act by manipulating the Location and Size properties (more specifically, the X, Y, Width and Height sub properties). Manipulating these settings is most helpful when trying to fine tune small incremental position and size changes.

Nudge Keys
Field position and field size can be adjusted in one pixel increments via the following nudge keys:

- Up arrow - Move field up vertically one pixel
- Down arrow - Move field down vertically one pixel
- Shift+Down arrow - Add one pixel of height to the bottom of the field
- Shift+Up - Remove one pixel of height from the bottom of the field

Delete A Field
To delete a field, select the field and click the Delete Field button on the button bar, clicking Field | Delete Field, or by selecting the field and pressing the Delete key.

Copy A Field
To copy a field, select the field you want to copy, then click the Copy Field button on the button bar, or click Field | Copy Field.

Activating A Field
Once a field is added, positioned and sized, you’re ready to activate the field so it can be accessed from a regwin. All field access properties are contained within the Accessibility, Appearance and TextFont properties described below. These properties are made available when you select a specific field. Before we get into the details of these properties, it’s important to determine whether you want your field to be accessible for extracts only or both extracts and pastes. This is important because there are significant differences in the way you configure your field’s properties based on which capability you want the field to support in your simulation.
Properties

Accessibility
Property grouping that contains the Name and Value properties.

Name
Name is the identification value you want to assign to your field. This name will appear in the field navigator combo box on the simulator’s button bar. It also serves as the accessible name value when referenced within the appspace editor.

Value
This is default value you want the field to have when you run the simulation. Unless changed during a paste, this is the value the appspace editor will return as the field’s accessible value.

Appearance
Property grouping that contains the EditStyle and TextBoxColors properties.

EditStyle
EditStyle allows you to specify whether your field will support extracts only or extracts and pastes. When you select the value Transparent, this tells the simulator that you want the overlay to be transparent in appearance and merely serve as a place holder for mapping in the appspace editor or appspace wizard, and the extraction of the default value set in the Accessibility | Value property. During a simulation, a transparent field will have no visual impact on the screen simulation but will allow itself to be accessed and mapped using the defined value. When set to Transparent, the TextboxColors and TextBoxFont properties have no effect on the simulation.

When you select TextBox, you’re letting the simulator know you want this field to alter the underlying screen’s image appearance with this overlaid field and make the field available for mapping where extracts and pastes are supported. While the field uses the value set in the Accessibility | Value property, it can be pasted to and thus overwritten with a new value during a simulation. Since the field has a visual impact on the screen image, the TextboxColors and TextBoxFonts properties do apply.

TextboxColors
The TextboxColors property allows you set both the background and text colors for your field. You can either set the colors manually by modifying the RGB codes (background, text color), or you can pick the
colors from the **Select Colors** dialog. To access the Select Colors dialog, click in the property value field and click the subsequent ellipsis. When the dialog loads, you have the ability to pick the background/foreground text combination that most closely matches the field you’re attempting to simulate.

### TextboxFont

Group of properties that affect the visual appearance of the overlaid field’s font. All subsequent properties can be set by clicking the property value field and clicking the subsequent ellipsis. When you do, you’re presented with the **Font** dialog which allows you set all the font properties.

#### Name

Name of the font you want to use. The simulator supports **TrueType** fonts only.

#### Size

Size allows you to set the font size based on the units defined in the **Unit** property.

#### Units

Units allows you to define the unit type upon which to base the font size. Valid values are: World, Pixel, Point, Inch, Document and Millimeter.

#### Bold

Bold determines whether or not the font is bolded.

#### GdiCharSet

GdiCharSet allows you to supply a byte value that defines which GDI character set should be used to render the font. A value of 1 (default), means use the default operating system character set will be used.

#### GdiVerticalFont

GdiVerticalFont determines if a font is compatible with native Win32 controls on non-Unicode platforms. Default is **False**.

#### Italic

Italic allows you to define whether or not you want the font to be italicized.
**Strikeout**
Strikeout allows you to define whether or not you want the font to be rendered in strikeout format.

**Underline**
Underline allows you to define whether or not you want the font to be underlined.

---

At any point during the creation of your simulation, you can test it by clicking the run button on the simulator button bar. When you do, the property grid will hide and the design surface will transition into simulation mode. If you've done your job right, the simulation should look just like the real application. To exit simulation mode, click the simulated application's titlebar and then press Ctrl+C. This will bring you back into design mode.

To save your simulation, either click the **Save** button on the simulator button bar, or click **File | Save As**, give your simulation a name and click the subsequent **Save** button.

Once you've completed configuring your simulation, you're now free to load it in simulation mode and create an appspace based on it using either the appspace editor or the appspace wizard where the overlaid and activated fields can be mapped as snippets and mapped to xmodels. For all intents and purposes, the simulated application should look and feel like the real application in terms of how it relates to Ratchet-X.
What is a “fingerprint” and how do I tie back to a Ratchet-X license?

Ratchet-X uses a flexible licensing mechanism to help ensure customers purchase only the number of licenses needed. Regardless of whether Ratchet-X is licensed on an installed seat, concurrent seat or SAS-basis, in all cases, usage is tracked and metered by RatchetSoft on a monthly basis. The cornerstone of this metering is the workstation fingerprint.

**Ratchet-X Workstation Fingerprint**

When Ratchet-X Commander is installed, a workstation ID is derived based on a number of non-user, non-customer specific criteria. This “hash”, uniquely identifies a given workstation and its associated a license. The fingerprint takes the form of a seven-to-ten digit code. For example, a fingerprint might look like the code; "1260737359". You can locate your workstation's fingerprint by loading Commander and clicking **Help|About**. When you do, you'll be presented with the Commander **About Ratchet-X Commander** dialog (see **Figure 1** below).

**Figure 1**
The fingerprint value is listed to the right of the label **Fingerprint**.
In order to protect user privacy, all data reported to RatchetSoft is linked to this fingerprint value only, which in or itself, has no inherent value. On its own, the fingerprint does not convey; Who the user is?, Which workstation Commander is running upon?, Which actions are being run?, etc. The only data conveyed to RatchetSoft with regard to usage is as follows:

- Fingerprint
- Number of actions run since the last load of Ratchet-X Commander (not which actions, or what these actions do)
- Version of Ratchet-X being run
- License with which this instance of Ratchet-X Commander is associated

All usage data is available upon request. If you request a usage report from RatchetSoft, you'll be provided a list of usage for requested specified period that displays the usage information listed above. In order to determine to which workstation a given usage log entree applies, load the About Ratchet-X Commander dialog and check the Fingerprint value in white box. Note that for security reasons, this is the only way to tie usage back to a given user's workstation or VM.
What Is The Sample Code Editor?

The Sample Code Editor is a code repository where battle-tested and proven code is stored for future use. These code snippets can be used during all aspects of creating a Ratchet-X integration and are part of our “Best Practices” for creating new Action code and Connector code. To open the Ratchet-X Sample Code Editor, find the following icon (see Figure 1):

**Figure 1 - Inside Action Editor**

![Figure 1 - Inside Action Editor](image1)

When you click that icon, you will get a dialog asking to download and run the Code Editor. Click Yes for these dialogs. Then you will see the Code Editor window (see Figure 3):

**Figure 3**

![Figure 3](image2)

```java
private void WaitForRegWin(RegWinInfo rw)
{
    int RWMaxWait = 15;
    WaitForRegWin(rw, RWMaxWait);
}
private void WaitForRegWin(RegWinInfo rw, int seconds)
{
    Log("Waiting for RegWin: " + rw.ID);
    bool found = Context.WaitForRegWin(rw, seconds);
    if (!found)
        throw new Exception("RegWin Time Out: " + rw.ID);
}
```
The Code Editor contains a list of the current library code snippets in the upper left pane, editing controls in the upper right pane, and the sample code in the bottom pane.

When you click on one of the code snippet titles in the upper left pane, the code will show up in the “Template” tab at the bottom (see Figure 3). There are code snippets to assist with handling a Login / Prompt User / Retry loop, a snippet for handling common data parsing methods used in Ratchet-X, some general Utility functions that are commonly used for cleaner code, to name a few.

There is also a more advanced scenario for the Code Editor, where you can auto-generate Action code from a pre-existing Appspace. For this example, let’s look at the “Basic Template” code snippet depicted in Figure 4.

Figure 4

You can see in the Template tab, a command shown called %merge:appspace %. This tells the Code Editor to use the DotLiquid templating engine for this code snippet. (To learn more about DotLiquid, see http://dotliquidmarkup.org/). When you click on the “Merged Code Sample” tab, you are prompted to select an existing appspace file. After selection, auto-generated code appears in the “Merged Code Sample” Tab, (see Figure 5) This boilerplate code can then be copied in to an action or connector, which saves time and minimizes coding errors.
Figure 5

If you would like to contribute your own code snippets, please contact support@ratchetsoft.com to become certified and receive a password for creating and editing the code snippets.
MyScreenTester

The MyScreenTester desktop utility allows you to quickly test your application screens to determine if they are viable candidates to integrate and automate using Ratchet-X. Once downloaded to your desktop, this utility allows you to test various screen regions within your application to determine if and which recognition technique can be used to extract data from your application screens. The utility guides you through the testing process so it is quite easy to use.

Download MyScreenTester Utility

Google Chrome Notes

MyScreenTester uses three different screen text extraction techniques, the first of which is Accessibility. This means any application running in accessibility mode can be queried and have its screen data reliably extracted without having to rely on screen scraping or screen OCR. However, this requires the application being queried to support accessibility. By default, Google Chrome DOES NOT run in accessibility mode. Fortunately, you can run Chrome in accessibility mode by taking the steps below. Note turning on accessibility mode will not impact Chrome's performance in any other way. Also note that if you decide not to run Chrome in accessibility mode, MyScreenTester will still be able to extract data from the screen using technique 3, OCR.

Running Chrome in Accessibility Mode

There are two ways to run Chrome in accessibility mode.

Method 1 - Turning on Accessibility Mode from within Chrome.
To turn on accessibility mode within Chrome, type `chrome://accessibility` in the address bar and hit Enter. This will bring up the Chrome Accessibility setting page (see below)
Next, click the **on** link located at the top to the right of the label **Global accessibility mode**.

**Method 2** - Command Line Parameter.
Load Chrome from a command prompt adding the accessibility parameter: `chrome.exe --force-renderer-accessibility`
RatchetSoft Knowledge Base

Project Methodology and Best Practices

Ratchet-X is an innovative desktop integration platform that allows you to integrate your existing software applications with web services, websites, electronic forms, databases and other desktop applications, without requiring you to make changes to your existing applications. Ratchet-X empowers by granting you greater control over the integration possibilities of all your applications.

While Ratchet-X is a powerful and versatile integration tool, it’s not a magic bullet that allows you to solve every integration challenge. This being the case, it’s important to understand where Ratchet-X fits best and how you should go about implementing the platform when it is a fit. This implementation methodology guide is designed to help you with this process.

The Ratchet-X Implementation Methodology is comprised of the following steps:

1. Evaluate opportunity
2. Complete RX Process Definition Document
3. Develop assets (appspaces, actions, xmodels)
4. Test
5. Deploy

1) Evaluate Opportunity

Most Ratchet-X implementations start with the desire to satisfy a tactical integration requirement. This requirement usually involves reducing the number of keystrokes associated with repetitive tasks, reducing data entry errors and/or the desire to compile a more comprehensive view of an entity or transaction. For example, an organization requiring all account managers to perform a credit check on new customers can use Ratchet-X to integrate its CRM or accounting system with a credit bureau’s reporting web service, web site or credit retrieval software.

Once Ratchet-X is in place, its application spreads like wildfire as users better understand its versatility and the number of less-obvious use cases to which it can be applied.

When evaluating tactical Ratchet-X opportunities, you should keep the following implementation considerations in mind:

- Ratchet-X should be used to facilitate repetitive interactions between applications. Occasional and one-off inquiries may not justify the effort of implementing Ratchet-X.
• The more data involved with extracts and pastes, the more applicable Ratchet-X tends to be. Since actions can automate screen flow and navigation, the degree to which fields are dispersed across multiple screens and applications should not be a major consideration.

• Data flowing between applications can be harmonized based on logic. In other words, if you can’t call a web service or paste data into your application because an irreconcilable data fidelity mismatch exists, Ratchet-X will provide little value. An example of a logic-based harmonization rule is the simple conversion of a US state from its full name to its two letter abbreviation.

• While at first glance, Ratchet-X appears to be user-directed integration tool its ability to perform batch integrations should not be underestimated. In fact, if the batch integration processing involves systems that do not possess an API or direct access to its data store, Ratchet-X may be the only way to perform the integration. For example, imagine an Accounts Payable department is scanning invoices and would like to match and update existing purchase orders in its accounting system with data from the scanned invoices. If the accounting system doesn’t have a robust API, automating the process becomes impossible and the manual process remains arduous and error prone. However, if the process is logical and thereby repeatable, you can use Ratchet-X Aggregation Workstation to automate the batch process.

For more on Ratchet-X use case patterns, we recommend viewing the RatchetSoft Video Series video; “Innovative Customer Uses of Ratchet-X”.

2) Complete RX Process Definition Document

Once you’ve determined Ratchet-X is a fit for your immediate integration requirement, the next step is to complete the RX Process Definition Document.

This document defines a methodical approach to analyzing and documenting your Ratchet-X integration. While by no means the only way to undertake a project, it is a time and market tested way for gathering requirements and defining specifications at a level of granularity required to get it right the first time and your integration in production as quickly as possible.

The RX Definition Document is comprised of the following sections:

• Project summary and requirements
• Desktop opening state
• Step analysis
• Desktop return state
• Custom logging requirements.
Project Summary and Requirements

This section allows you to supply summary information for the project such as: project name, author, date, version, etc. It also allows you define which data sources are being integrated in this project along with any desktop support requirements such as operating systems and browser versions.

Desktop Opening State

This sections allows you to define the state of the desktop at the point Ratchet-X “initiates involvement”. Initiates involvement means the first time a magic button appears in an integrated application or the point at which an action is executed via any supported means (file drag drop, command line, service invocation, etc). Key facts to consider when completing this section include the following:

- Which applications are loaded (which screens are being displayed)?
- Is the user logged into these applications or will system logins be required during the process? Do logins expire on a regular basis and should the action take this into consideration?
- What’s the trigger event that initiates Ratchet-X’s involvement (e.g. the finding of a regwin and display of the magic button in one of the loaded applications, the receipt of an email, notification from an application, etc.)
- If the action is triggered by the magic button, what should the alert text associated with the action be? Should there be a magic button override?

Step Analysis

The step analysis is the heart and soul of your Ratchet-X integration. The step analysis defines in a detailed manner:

- Which applications and data sources need to be integrated?
- Which UI elements, fields or services need to be integrated?
- What navigational operations must be taken to get the applications and data sources into the proper state for processing?
- What is the data structure and attributes of the data being exchanged (think xmodel)?
- How must the data be harmonized and manipulated in order for the integrated systems to exchange data?
- Which, if any, message screens may be encountered during the process?
- Will information need to be collected from the user during the process?
- How should errors be handled?
- Do different user types experience different application experiences (different screens, hidden fields, disabled buttons, etc)? Does the application operating in multiple modes?
The step analysis should serve as pseudo code for your action. It’s critical to be as precise as possible so that all integration conditions can be accounted for. In fact, it’s considered a best practice to assign someone who is NOT the actual end user to build the step analysis. While primary user input is critical to defining the process, users who perform these processes every day tend to operate from “muscle memory”, and often omit exception states. Moreover, they tend to merely focus on the way they specifically perform a function versus how the integrated applications and data sources behave. So while the primary user should be an important party to the effort, he should not be the person who actually defines the step analysis.

**Desktop Return State**

The desktop return state describes the state the user’s desktop should be left in at the conclusion of performing the aforementioned steps. Things to consider include:

- Load state of applications and navigation points.
- Location of applications windows on the desktop.
- Logins and logouts.
- How return state tees up follow-on transactions either performed by subsequent Ratchet-X actions or manual transactions carried out by the user.
- Reset of any operating system level display configurations (DPI, font smoothing, gray scaling, etc.)

**Custom Logging Requirements**

By default, Ratchet-X captures a plethora of events related to regwins and actions. However, from time-to-time, a project may call for capturing runtime-related transaction data such as; “what data was processed by an action?” or “how was data used in a transaction validated and/or transformed?” Further, the use case may call for logging data to be sent to a specific logging queue or analytics tool. All custom logging requirements should be documented in this section.

**3) Develop Assets**

Once your RX Process Definition Document is fully attributed, it’s time to develop the assets. There is no hard and fast rule with regard to the order in which you develop the assets. When RatchetSoft’s Professional Services Group is engaged in a project, we tend to develop the actions and xmodels first using hard coded test data and then building the appspaces. The only reason you might lean towards developing the action first is because during the course of creating the action, you’ll also design and create your xmodels which in turn can be immediately used within the appspaces. However, you can always create your appspace first and backfill with xnodelemappings once the xmodels are available. It’s really up to you and the way you want to work.
4) Test
Once you’ve created your assets, you’re now ready to perform end-to-end testing. Keep the following points in mind with regard to testing:

- We strongly recommend you perform testing with Commander and the actual applications before rolling out your assets into production. Test harness are great for alpha testing, however, there is no substitute for how applications behave in production. Also, it’s often the case that certain message screens are not discovered until testing with the applications is performed.
- Test appspaces in all target runtime environments. If you’re supporting multiple browser or application versions and operating systems, make sure you test your appspaces with each plausible combination.
- It is often the case that testing a Ratchet-X integration may require more time than it took to design and build the original integration. This is normal so be sure to properly account for it in your project plan.
- Make sure you have plenty of test data on hand. Ratchet-X Professional Services projects require a minimum of ten test records for each use case.

Your project is best served adhering to the following iterative test process:

1) Perform unit and alpha testing on the dedicated development workstation.
2) Once unit and alpha testing are complete, you’ll need to perform the critically important step of variance testing. Variance testing is the process by which you create a deployment package using the Package Editor and deploy it to a subset of users that represent the kind of environmental variance you anticipate encountering in production (variance with regard to screen resolution, user types, operating system and browser versions, etc). We recommend 3-5 workstations as a minimum.
3) Any issues uncovered during variance testing should refactored into the assets resulting in the creation of a new package and another round of variance testing. You should iterate through steps 2 and 3 as many times as required. It’s not uncommon for variance testing to uncover significant undocumented requirements or runtime conditions. This is to be expected.
5) Deploy
Once you’ve developed and tested your Ratchet-X assets, deploying the solution is quite easy. Ratchet-X should be deployed in accordance with Rathced-X deployment best practices. For more information on deploying Ratchet-X, click here.

6) Updating the RX Process Definition Document
Though it’s tempting to get caught up in the jubilation that is a successful Ratchet-X implementation, it’s important you update the RX Process Definition Document based on findings and modifications made during the development and testing phases. While it is strongly recommended the updating of the document occur in real time as changes are made, it can be done at the end of the process once the integration is deployed. Either way, make sure it gets done.

Note To Third Party Providers
One of the most powerful aspects of the Ratchet-X platform is it allows people to create appspaces and actions independently of each other. These assets are then linked together at runtime by the platform through the xmodel abstraction layer. This being the case, it’s possible to create actions for general use that can be consumed by thousands or millions of users over time – assuming your action leverages publicly available xmodels that appspace authors can reference in their appspaces. This powerful capability is fostering a community of third party providers who are creating Ratchet-X actions to front end their service and transaction-based offerings. If you’re one of these providers, you should note that certain aspects of this methodology may be different in your case. For example, if you’re creating actions for general consumption, you’re not necessarily concerned with making sure they work with specific appspaces so there is no need to create an application screen specification or create appspaces.

For more on creating Ratchet-X actions as a third party provider, contact us partners@ratchetsoft.com.