

Use of Assistive Technology to Augment API Capabilities

Hume L. Peabody[1]
NASA-Goddard Space Flight Center, Greenbelt, MD 20771

and

Kiet Vu[2]
Aerodyne Industires, Cape Canaveral, FL 32920

Application Programming Interfaces (APIs) allow for access to data and capabilities of computer applications by developers or users with experience in computer programming. Recent development with both Thermal Desktop and ESATAN-TMS have provided APIs to allow users to develop their own capabilities that interface with the Graphical User Interfaces (GUI) or manipulate the thermal model data. However, these APIs are only as good as the breadth of features in the native code accessible through the API. If a particular code's feature is not accessible through the API, then users have very limited options besides waiting for updates to the API that expose the necessary functionality, particularly if model data access or user action, such as a button click, is required. However, Assistive Technology features that allow for users with a disability to more fully experience a software's capabilities may be creatively utilized to gain further access to data and capabilities not yet exposed by the API. This paper describes the process to augment the features of the OpenTD API via assistive technology and describes how to identify the application instance, navigate GUI elements, updates values on forms, and execute actions such as selecting a listbox item or clicking a button. It concludes with identifying some of the pitfalls to avoid and describes methods to best implement this approach.

Acronyms and Nomenclature

ASCII	=	American Standard Code for Information Interchange
API	=	Application Programming Interface
CAD	=	Computer Aided Design
GUI	=	Graphical User Interface
FEM	=	Finite Element Model
IM	=	Integrated Modeling
RST	=	Roman Space Telescope
STOP	=	Structural Thermal Optical Performance

[1] Staff Thermal Engineer, Mail Stop 523, Goddard Space Flight Center, Greenbelt MD 20771.
[2] Junior Mechanical Engineer, Aerodyne Industries, Cape Canaveral FL 32920.

I. Introduction

Application Programming Interfaces (APIs) have been introduced in the last few years for tools to perform thermal analysis for space applications, such as Thermal Desktop and ESATAN-TMS. These interfaces have allowed engineers and developers greater access to the model data and/or capabilities of the code to enable the development of additional utilities and features by users and enhance the overall capabilities of the destination tool. For ESATAN-TMS, the underlying language and the ASCII nature of the input files did allow for scripting to be used to automate some portions of the thermal analysis process. For Thermal Desktop, which contains most of the model data in a proprietary binary file, however such scripting was very limited to batch file commands and did not allow for much access to the underlying data structures that defined the thermal components. With the inclusion of the APIs, some users have already began producing tools and utilities to take advantage of the new features. For the Roman Space Telescope project, tools have been developed to capture an image viewpoint and temperature scale, apply the mapping interpolation factors using multiple output files, and to re-create that image viewpoint with mapped temperatures for both the thermal model and the structural Finite Element Model (FEM) and export the image to a folder or to a PowerPoint presentation. Furthermore, for the Habitable Worlds Observatory project, a utility was developed to extract the mass for specified Domain Tag Sets and to apply a correction factor to achieve the desired mass based on inputs from CAD or FEM models.

However, the ability to interface with the thermal analysis code is limited to what methods and properties have been developed by the vendor and exposed in the API and likely do not encompass the full set of features available through the Graphical User Interface (GUI). If a particular feature is only available through the user interface, a developer has little choice but to request that the capability be added to the API in subsequent releases and wait for the vendor to accommodate.

That said, there is a method recently identified that allows programmatic access to elements of a GUI, such as text boxes, buttons, listboxes, etc. through the use of Assistive Technology. Assistive Technologies are algorithms and libraries that interact through the operating system to identify windows and controls in computer applications, mainly for the purposes of making the usage of these application easier for users with a disability. For example, a user with motor coordination challenges may dictate into a microphone to type a letter in a Word Processing program that was not originally developed to have such a feature. This is accomplished by an intermediate program accessing the GUI control and populating it with the data provided by the user in a means different than expected (i.e. dictation instead of through typing on a keyboard). This paper highlights how this approach was utilized to access GUI features in Thermal Desktop using a combination of the OpenTD API and the Assistive Technology libraries in the .NET framework using Visual Studio with the Visual Basic compiler.

This paper seeks to provide some step-by-step instructions for navigating the Windows hierarchy to find particular controls and retrieve data or execute their actions. While the code snippets provided herein are somewhat laborious, they do seek to provide a road map for developers who may be interested in utilizing this capability. In the absence of having control over the source code of a particular commercial product, this may be the only way for external developers to implement their goals for tool development without the need for vendor updates.

II. Motivation

The Roman Space Telescope project features Integrated Modeling (IM) as a core aspect for validating requirements that are impractical or impossible to verify during ground testing. A major component of the RST IM is the Structural Thermal Optical Performance (STOP) modeling, which maps the predicted temperature distribution throughout the spacecraft onto the structural FEM. The mapped temperatures are then used to predict the distortion of the structure due to thermal expansion or contraction from ground temperature conditions. Lastly, these distortions are then transferred to the optical model in terms of figure distortion and rigid body distortion to evaluate optical and pointing requirements. For a flagship mission like RST, this involves very large and complex models and requires careful attention to modeling approaches in both thermal models and structural FEMs to ensure that temperatures can be accurately mapped.

An important step between temperature mapping and use of mapped temperatures is verification. Unfortunately, this is a time consuming and difficult step at present as there is no readily identifiable mathematical metric that can be used to determine the “goodness” of the mapping. Therefore, analysts are left with few options but to manually compare contour maps between the thermal models and structural FEMs. But even the generation of these images itself is a very time-consuming process if done manually. Recent efforts on RST[1] have developed tools that help to automate the image generation process with contour plots from both the thermal model in Thermal Desktop® and the structural model in FEMAP®, using the same color bar values and color shades as well as the same viewpoint and

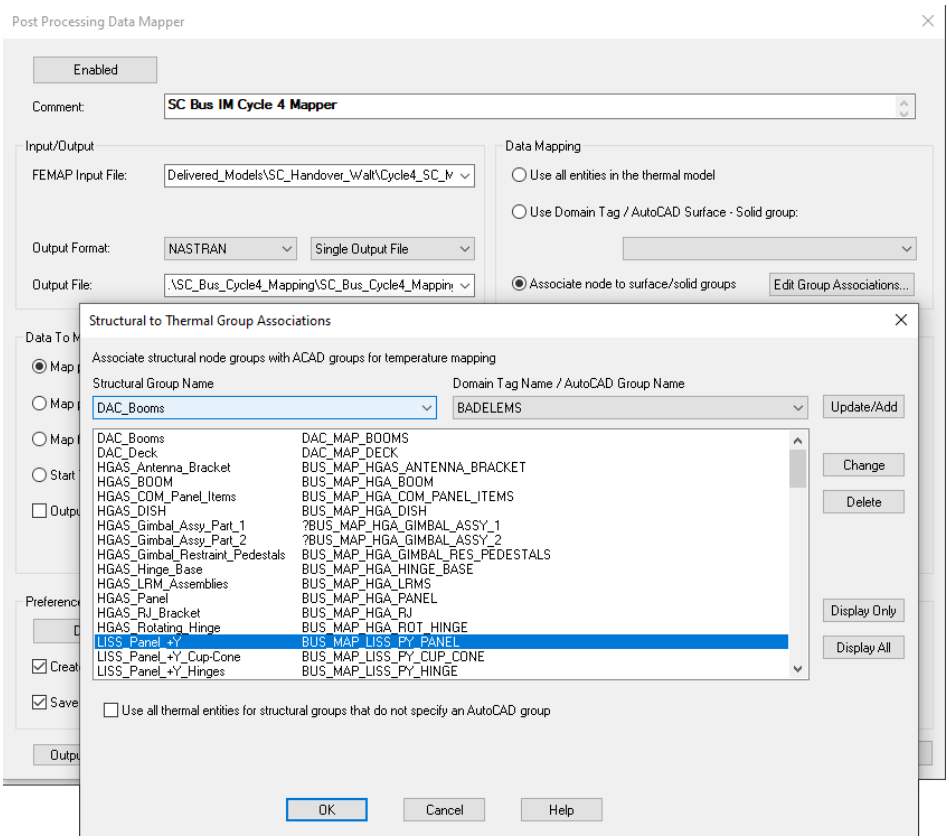

Figure 1. RST developed an application using Automation Elements to interact with the Thermal Desktop GUI: *Use of assistive technology enabled capabilities to be developed that were not available through the OpenTD API at the time.*

scale factors for the model display. This has reduced days of tedious work by structural and thermal analysts down to an overnight click of a button to generate about 500 images. But this capability was only enabled by the use of the Assistive Technologies described herein.

Using Thermal Desktop for the mapping allows analysts to specify Group Associations (Figure 1) between parts of the thermal model and corresponding parts of the structural FEM.

The Thermal Desktop GUI allow for the display of one or more of these groups at a time by clicking the "Display Only" button with the selected groups in the listbox, rather than displaying the ent ire FEM. However, at the time of this writing, the OpenTD API did not include methods to execute these features which were realistically only available through the GUI. Examining the methods available for the DataMapper object in the OpenTD API (Figure 2) shows that there are no corresponding methods to execute the "Display Only" or "Display All" features available in the Thermal Desktop GUI.

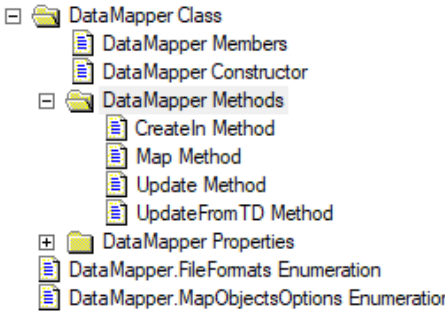

Figure 2. Methods Available in the DataMapper class in Open TD: *No method exists for Displaying the FEM*

However, through the use of Assitive Technology, these features can be accessed and executed. The remainder of this paper seeks to explain how to query the control hierarchy in Windows applications to find the particular control of interest as well as to retrieve or assign data to is as well as to execute its function.

III. Identifying the GUI Elements

The first challenge to using the Assistive Technology is to be able to identify the relevant windows and elements with which to interact. A tool like *inspect.exe*, which can be installed as part of Visual Studio by including the

Windows 11 SDK, allows users to actively query open windows for identifying characteristics, such as: Name, ControlType, InvokePatternAvailable, etc. Figure 3 (Top) shows the results of hovering over the red X close button on an instance of Notepad in the inspect.exe tool. Figure 3 (left) shows that during this session, multiple programs were open in Windows (represented at the top level as the Desktop 1 pane), including the Inspect tool, the TaskBar, Microsoft Visual Studio, as well as an instance of Notepad. Furthermore, Figure 3 (left) shows the path from the Desktop through the Notepad window to the "title bar" and finally to the "Close" button. The right listbox shows the various properties of the selected control that can later be accessed by the Assistive Technology libraries. Furthermore, it also shows the further controls the comprise the Notepad program, including the Close button. Understanding this hierarchy of controls is crucial to being able to find a particular control within an application of

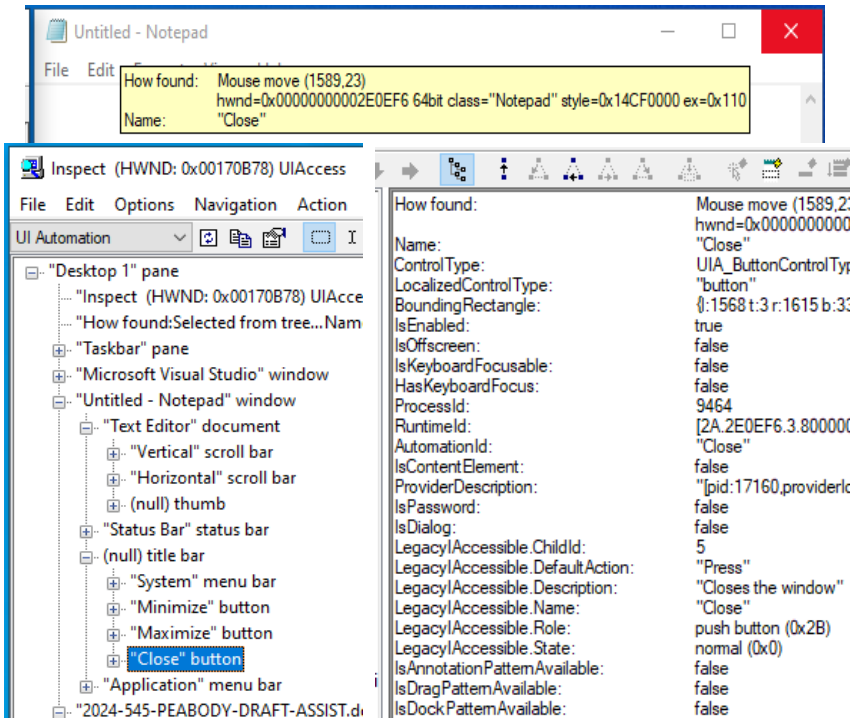

Figure 3. Results from Inspect.exe hovering over the Close Button on the Title Bar of a Notepad instance: *(Top) shows user hovering over Close button of a Notepad instance, (Left) Shows the hierarchy in Windows of the identified control, (Right) Shows all the properties of the identified control*

interest to the developer.

IV. Interacting with the Elements via Code

The second step to utilize Assistive Technology in an application being developed is to ensure that the required references are included in the project. To allow interaction with elements in external programs using the .NET framework, the UIAutomationClient and UIAutomationTypes references must be included. These libraries, then allow developers access to objects in the .NET framework that can query controls found by navigating the element tree in Windows based on expected names and control types identified using the inspect utility, described in the previous section.

To find a particular component, such as a Window with the title "Notepad", it is generally best to begin at the RootElement and search for the PropertyCondition type that is desired, such as a "window". The Visual Basic code below will find the first "window" starting from the RootElement, which is generally the last application opened in Windows.

```
Dim AEWindow as AutomationElement = AutomationElement.RootElement.FindFirst(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.LocalizedControlTypeProperty, "window")
) '// Find the first element in the Windows environment
```

The TreeWalker object can then be used to find the next sibling window and loop until the desired window is found or until all windows are checked and no more siblings can be checked. The code below will find the first instance of Notepad.exe (if open) and exit the loop or exit the loop if no instance is found.

```
Do While Not IsNothing(AEWindow)
  If AEWindow.Current.Name.ToUpper.Contains("NOTEPAD") Then Exit do
  AEWindow = TreeWalker.ControlViewWalker.GetNextSibling(AEWindow)
Loop  '// Loop through all elements in the Windows environment until we find NOTEPAD in the name, then exit
```

Once the desired window is found, a similar process can be applied to find a desired control, such as a button, textbox, listbox, etc in that window. In this case, the TitleBar of the Notepad window is sought. After that, the Close button on the TitleBar is sought next.

```
Dim AETitle as AutomationElement = AEWindow.FindFirst(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.TitleBar)
) '// From the NOTEPAD window we found earlier, find the TitleBar control
Dim AEClose as AutomationElement = AETitle.FindFirst(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.NameProperty, "Close")
) '// From the NOTEPAD TitleBar control we found earlier, find the Close Button
```

Once the desired control (button in this case) is found in the Notepad window, it is expected that some action would be taken utilizing the button control. This is accomplished using Control Patterns, such as InvokePattern to click a button or TextPattern to access textual information. The code below programmatically "clicks" the Close button window of the TitleBar of the Notepad window to terminate the program.

```
Dim IPClose As InvokePattern = AEClose.GetCurrentPattern(InvokePattern.Pattern)
IPClose.Invoke() '// From the Close Button of the TitleBar in NOTEPAD, execute a button click
```

The above code may seem simplistic and may not seem all that useful at first glance; after all, it is simple for a user to simply close an instance of a program themselves. However, the particular application for which this was developed was a function that computes the thermal mass from Thermal Desktop for a selection of objects. Thermal Desktop includes a command to do this which can be executed through the OpenTD API for a given selection set. The function would be called for each group of objects of interest, which may number from tens to the hundreds of objects in a realistic model. However, each time this command is called, it opens the resulting values in a Notepad window, which would have then resulted in tens to hundreds of Notepad instances being opened. Being able to programmatically close each instance after opening resulted in a much more usable capability for the user without the need to close all the Notepad instances manually. It also illustrates the ability to reach in to another program to execute a capability or to retrieve data. This capability is at the heart of this paper to illustrate methods by which to expand on the OpenTD API capabilities to access data and execute commands that have not yet been exposed through the API itself. It should be noted however, that this capability is not limited to the Thermal Desktop GUI but could be used for any application where the AutomationElements (or Assistive Technology capabilities) are available.

This approach represents a work-around when a particular GUI capability is not available through other means, be it command lines or methods in an API. But in the absence of these, this approach is viable, even if not desired, until such time as the features are available through other means. It should be noted that this approach does come with risks which are discussed later.

V. Accessing Data in a Control

Being able to execute a button click is useful for some cases. But it is more likely that being able to access and manipulate data is even more beneficial. Another capability that was not available using the OpenTDv63 API was the ability to retrieve or set the Max and Min values for the ColorBar in the Thermal Desktop Edit Color Bars form as shown in Figure 4.

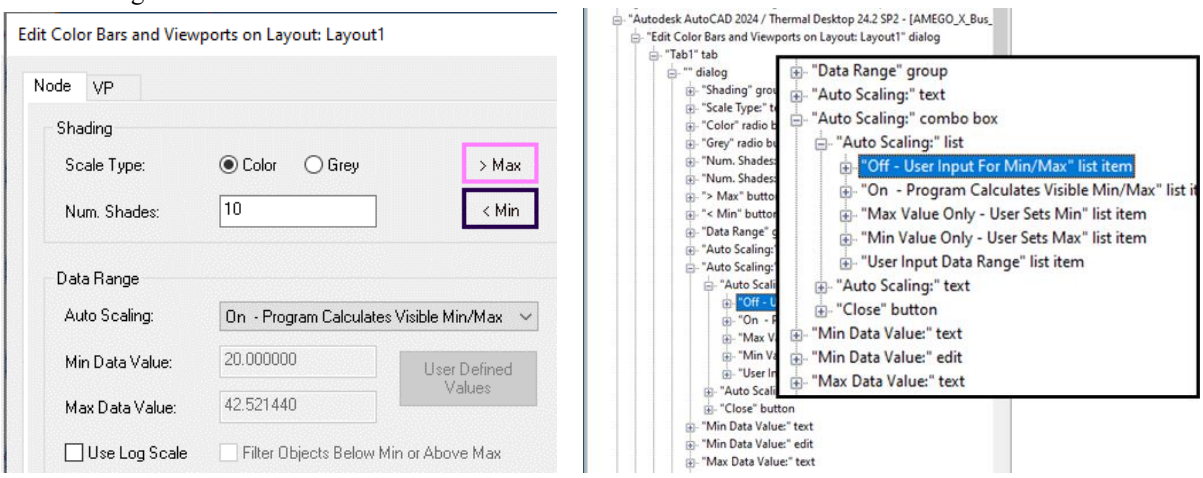

Figure 4. Results from Inspect.exe for the Edit ColorBar Form in Thermal Desktop: *(Left) Shows a portion of the form in Thermal Desktop, while (Right) shows all the AutomationElements on the form. Note that many are not uniquely named (e.g. there is a Min Data Value text end edit)*

Similar to the process described in Section IV, the Min Data Value and Max Data Value text boxes were determined by walking the hierarchy from Window to Form to Control. Assuming the variable AEMinValue holds the Min Data Value textbox control, the following code using the TextPattern type can be utilized to retrieve the value, where the 1000 argument is the maximum number of characters to retrieve.

```
Dim TPMinValue As TextPattern = AEMinValue.GetCurrentPattern(TextPattern.Pattern)
Dim StrMinValue as String = TPMinValue.DocumentRange.GetText(1000) '// Get the Min Data Value
```

Similarly, the ValuePattern type can be used to set the text of the Min Data Value text box.

```
Dim VPMinValue As ValuePattern = AEMinValue.GetCurrentPattern(ValuePattern.Pattern)
VPMinValue.SetValue("0.0") ' // Set the Min Data Value
```

It should be noted in Figure 4, that the Min Data Value and Max Data Value textboxes are disabled due to the Auto Scaling being set to a value where the program calculates the Max/Min. Therefore, the above SetValue would not be allowed on a disabled control. However, just as a button can be "clicked" programmatically, a ComboBox can have one of its list items also selected via code. To do this, it is necessary to determine the AutomationElement for the ComboBox and then find the list items associated with the ComboBox and lastly to execute the pattern to select the desired list item. The following code does that starting from the root element and tracing through the Thermal Desktop AutoCAD window to the ColorBar form, to the Node Tab to the AutoScaling ComboBox and lastly to the items in the Auto Scaling ComboBox. The first of these items is then selected using a SelectedItem pattern. One new concept is introduced in this code and is highlighted in red in the following code snippet. Instead of retrieving a single AutomationElement using FindFirst, a collection of AutomationElements is retrieved using FindAll. To access the individual AutomationElements within the collection, the index must be specified. Since the first item in the list is desired to be selected, index 0 is used in the following code snippet.

```
Dim AEAcad As AutomationElement = AutomationElement.RootElement.FindFirst(
    TreeScope.Children,
    New PropertyCondition(AutomationElement.LocalizedControlTypeProperty, "window")
) '// Find the first element in the Windows environment

Do While Not IsNothing(AEAcad)
  If AEAcad.Current.Name.ToUpper.Contains("THERMAL DESKTOP") Then Exit Do
  AEAcad = TreeWalker.ControlViewWalker.GetNextSibling(AEAcad)
Loop '// Loop through elements in the Windows environment until we find Thermal Desktop
If IsNothing(AEAcad) Then Exit Sub

Dim AEColorBarForm As AutomationElement = AEAcad.FindFirst(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.LocalizedControlTypeProperty, "Dialog")
) ' // Get ColorBar Form from Thermal Desktop Element

Dim AEColorBarTab As AutomationElement = AEColorBarForm.FindFirst(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.NameProperty, "Tab1")
) ' // Get ColorBarTab from ColorBarForm

Dim AEColorBarCombo As AutomationElement = AEColorBarTab.FindFirst(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.LocalizedControlTypeProperty, "combo box")
) ' // Get ColorBarComboBox from ColorBarTab

Dim AEColorBarComboList As AutomationElement = AEColorBarCombo.FindFirst(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.LocalizedControlTypeProperty, "list")
) ' // Get ColorBarComboListBox from ColorBarComboBox

Dim AEColorBarComboListItems As AutomationElementCollection = AEColorBarComboList.FindAll(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.LocalizedControlTypeProperty, "list item",
  PropertyConditionFlags.IgnoreCase)
) ' // Get ColorBarComboListItems collection from ColorBarComboListBox

Dim SIPColorBarComboListItems As SelectionItemPattern =
  AEColorBarComboListItems(0).GetCurrentPattern(
    SelectionItemPattern.Pattern
) ' // Set SelectedItem pattern for first ComboBox list item

SIPColorBarComboListItems.Select() ' // Invoke selection
```

Note that after this code is executed, the ColorBar form looks like Figure 5 (left). If the previous code block is executed, the Min Data Value textbox would be updated to 0.0, as shown in Figure 5 (right), having be enabled by selecting the "Off – User Input For Max/Min" list item in the Auto Scaling ComboBox.

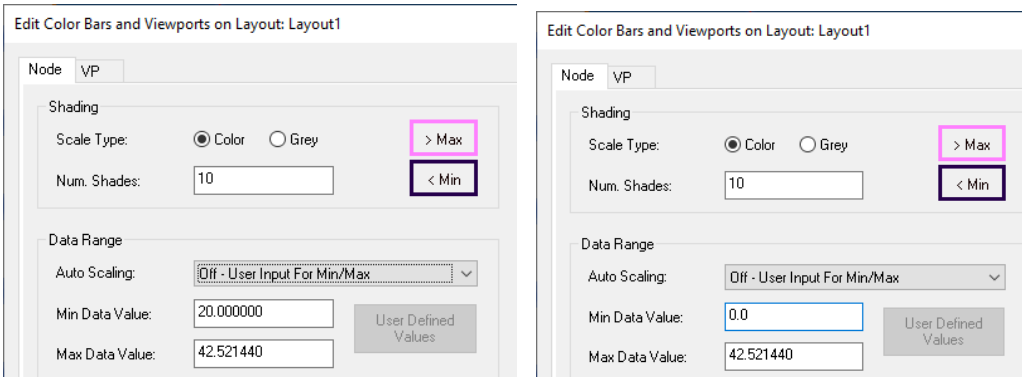

Figure 1. Results from using AutomationElements to Adjust ColorBar Form Controls: *(Left) shows the form after the Auto Scaling list item is selected, which enables the Min Data Value and Max Data Value text boxes, and (Right) shows the updating of the Min Data Value to 0.0 via AutomationElements*

VI. Finding Desired AutomationElements More Efficiently

Navigating the hierarchy of the automation elements is the safest method, but may be rather inefficient in the number of lines of code needed to drill down to a desired AutomationElement. Fortunately, a method exists to search through all descendants and not just the immediate children. Changing the TreeScope from Children to Subtree will search all the descendants instead of just the first generation of Children. Therefore, the code above can be reduced greatly by starting the search from the AutoCAD AutomationElement as shown below. While it may even be possible to start from the Root Element, this is not advised as each progressive level higher can greatly increase the number of checks of valid AutomationElements to see if they meet the PropertyCondition. A general recommendation is to start the search no higher than the desired application, in this case AutoCAD/Thermal Desktop.

```
Dim AEAcad As AutomationElement = AutomationElement.RootElement.FindFirst(
  TreeScope.Children,
  New PropertyCondition(AutomationElement.LocalizedControlTypeProperty,
  "window")
 ) '// Find the first element in the Windows environment

Do While Not IsNothing(AEAcad)
 If AEAcad.Current.Name.ToUpper.Contains("THERMAL DESKTOP") Then Exit Do
 AEAcad = TreeWalker.ControlViewWalker.GetNextSibling(AEAcad)
Loop '// Loop through elements in the Windows environment until we find Thermal Desktop

If IsNothing(AEAcad) Then Exit Sub

Dim AEColorBarComboListItem As AutomationElement = AEAcad.FindFirst(
  TreeScope.Subtree,
  New PropertyCondition(AutomationElement.NameProperty, "Off – User Input For Min/Max")
 ) ' // Search hierarchy for elements with a name of Off – User Input for Min/Max

Dim SIPColorBarComboListItems As SelectionItemPattern =
 AEColorBarComboListItem.GetCurrentPattern(
  SelectionItemPattern.Pattern
 ) ' // Set SelectedItem pattern for first ComboBox list item

SIPColorBarComboListItems.Select() ' // Invoke selection
```

The above code works as there is only one AutomationElement with the name “Off – User Input For Min/Max”, which is very specific. However, there is no guarantee that the NameProperty is unique. Therefore, it is advised if using this approach to provide further criteria via an AndCondition instead of a simple PropertyCondition. The AndCondition takes multiple PropertyConditions that must all be satisfied for the AutomationElement to meet the PropertyCondition. Likewise, there is also an OrCondition, which would be best used with a FindAll method and then looping through the resulting collection to narrow the AutomationElements of interest. If the code below is executed, it will return a collection with 2 elements, one for the text box (edit) and one for the label (text):

```
Dim MinValues As AutomationElementCollection =
  AEAcad.FindAll(TreeScope.Subtree,
  New PropertyCondition(AutomationElement.NameProperty, "Min Data Value:")
) ‘ // Search hierarchy for elements with a name of Min Data Value:
```

However, adding an AndCondition as below would return only the text box. The number of returned AutomationElements in a collection should always be checked using the .Count property

```
Dim MinValues As AutomationElementCollection =
  AEAcad.FindAll(TreeScope.Subtree,
  New AndCondition(
    New PropertyCondition(AutomationElement.NameProperty, "Min Data Value:"),
    New PropertyCondition(AutomationElement.LocalizedControlTypeProperty, "edit")
    )
) ‘ // Search hierarchy for elements with a name of Min Data value: that are also a TextBox
```

VII. Challenges with using Assistive Technology

While the capability to reach into a third party’s software GUI and interact with the data using Assistive Technology is quite powerful, it does come with some caveats and pitfalls that should be considered. Since any code developed using this technology interacts with the GUI itself, users should be careful to not interrupt the code with their own usage of other applications. For example, trying to select a list item in a listbox requires that the listbox has the current focus. Clicking on another control in another application may interfere with this. Therefore, these kinds of applications are best developed as longer term, fire-and-forget type programs best run while no other applications require user input.

Furthermore, the code snippets listed above do not include any error checking and trapping. It would be critical to check for any errors (such as Nothing being returned for a given AutomationElement). Similarly, before updating the value of any AutomationElement, its enabled state should be checked to ensure that such actions are legal. When using collections, it may also be necessary to loop through the items in the collection to ensure the correct AutomationElement is being manipulated. For example, a user might have multiple instances of Thermal Desktop open and the first may not be the intended instance upon which operations are intended. Further narrowing of the criteria for any AutomationElement is crucial to ensure things work as intended.

Lastly, development of code that interfaces with another application’s GUI may not work at a later time if the application’s GUI is changed by the developer. For example, a collection of radio buttons may be replaced by a collection of check boxes, thereby causing the developed application to fail. While this is an acknowledged risk, careful error trapping and checking can help to mitigate this. Furthermore, many mature applications generally do not make too many subtractions from their GUIs, generally preferring to add to capabilities rather than remove them. That said, any development that relies on another application’s GUI must accept this risk. It is judged that the benefits of utilizing this approach will generally outweigh the drawbacks.

VIII. Conclusions and Path Forward

APIs for thermal analysis codes have opened a path for thermal engineers with programming skills to be able to further access model and capabilities of the application programmatically, allowing for the development of tools and utilities to automate or further manipulate models. But these APIs only provide access to what the application developers have granted. However, the use of AutomationElements through Assistive Technology libraries allows the user access to data and capabilities beyond those provided by the API.

This paper illustrated the process for navigating the RootElement and all its dependents to find, modify, or execute capabilities currently reserved for use only in the GUI and not through the API. The search process begins at the Root Element, and by narrowing the scope (Children, Parent, Subtree, etc) and the PropertyCondition (NameProperty, Localized-ControlTypeProperty, etc.), specific AutomationElements can be identified. The use of Patterns (InvokePattern, TextPattern, ValuePattern, SelectionItemPattern, etc), allows the actions associated with these controls to be executed programmatically. This does however require careful narrowing of the parameters to ensure that the intended AutomationElement is properly identified as these properties may not necessarily be unique.

Usage of these techniques is best employed for applications that can be left running without interruption for extended periods of time without the need for user interaction. Furthermore, they might also be best served to be run on a computer that is left alone while the application is running if the application has frequent and major interactions with another application's GUI to prevent any conflicts. This technique was recently employed for the Roman Space Telescope to develop an application[1] which loops through each Domain Tag Set to Structural group mapping association and "executes" the Display Only button click in the Thermal Desktop GUI to show only the selected group (Figure 1). The resulting thermal model screen image was then captured and imported into PowerPoint. Subsequently, the corresponding image was also captured for the specified FEM group and imported onto the same slide in PowerPoint. This saved hours of manual labor to accomplish the same thing and was used in an overnight capacity to generate hundreds of images for comparison of the mapping. This capability was only enabled by usage of the AutomationElements as the Display Only feature was not available in the OpenTD API version available at the time of development.

The use of the OpenTD API in conjunction with AutomationElements has allowed for the development of tools and utilities to greatly reduce the manual labor associated with thermal mapping to structural models for thermal distortion and Structural-Thermal-Optical-Performance analysis. Opportunities to further take advantage of these capabilities will continue to be explored in hopes of further improvements in thermal analysis capabilities and reduction in the time required to complete the analysis.

Acknowledgments

The primary author would like to thank the Roman Space Telescope project for continuing to support the exploration of the OpenTD API to improve our thermal analysis processes and capabilities as well as Kiet Vu for proposing, exploring, and implementing this approach in the development of tools to improve our processes for the Roman Space Telescope.

Addendum

Trade names and trademarks are used in this paper for identification only. Their usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

References

[1] Peabody, H., McDonald, C., and Vu, K., "Improvement in the Thermal-to-Structural Model Mapping Process for Integrated Modeling for the Roman Space Telescope" in *55th International Conference on Environmental Systems*, Puerto Rico, United States, ICES-2026-588