Chapter 2

Designing the Logical Hierarchy

Imagine that you need to use WordPad, and you need to access it from the Start menu. How would you open the menu if you couldn't see the screen? How would you get to the application among the different items in the menu? How would you know where you were in the menu and what item your keyboard focus was on? By thinking about these questions, you have put yourself in the shoes of some of your users who need a way to navigate and interact with your UI.

Unlike users who can use a mouse and monitor to navigate the UI, users who use a screen reader primarily use a keyboard for navigating through the UI and audio devices to listen to where they are in the UI. It is, therefore, extremely important that the navigation and structure of the UI be useful, accurate, and logical. The following steps during the design phase will help to ensure that your product provides such structure and navigation:

1. Design what your UI should look like and how it will operate. The navigation and programmatic access of the UI should closely match its visual counterpart. If you make changes to the visual design, then you will need to make changes to the application's navigation and programmatic access as well.
2. Determine which UI framework you are going to use. Each framework has a different set of controls, flexibilities, and accessibility support. Depending on your UI scenarios, a particular choice may work better or worse. Take time to assess your scenarios with the framework's accessibility support. You may end up with painful costs because of your ignorance about the framework's limitations.
3. Identify the controls to create the UI. Use framework controls whenever possible and not custom controls. When using framework controls, use them as they were intended. Any irregular or nonstandard use of a control often leads to bad usability and accessibility.
4. After studying the logic of your navigation and the structure of your UI, design a *logical hierarchy*, which will enable you to plan out the accessibility in your product. An accessible solution is only possible when you fully understand the logic and structure of your own UI.
5. Plan for UI Automation (UIA) for any of your custom controls identified in step 3, including those custom controls based on framework controls. Remember that creating new custom controls is extremely costly. If you have no custom controls, you can skip this step.

In this chapter, we focus on step 4, how to design a logical hierarchy for your UI, and the next chapter walks through step 5 in detail. Both chapters may provide you with helpful information for steps 1 through 3, which may be part of the business planning and investigation of your application.

The Logical Hierarchy

What do we mean by the term "logical hierarchy?" When AT programs, such as screen readers, read your UI, visual presentation is not sufficient; you must provide a programmatic alternative that makes sense structurally to the users. A logical hierarchy can help you do that. It is a way of studying the layout of your UI and structuring each element so that users can understand it. A logical hierarchy is mainly used:

1. To provide programs context for the logical (reading) order of the elements in the UI.
2. To identify clear boundaries between custom controls and standard controls in the UI.
3. To determine how pieces of the UI interact together.

A logical hierarchy is a great way to address any potential usability issues. If you cannot structure the UI in a relatively simple manner, you may have problems with usability in your UI. A logical representation of a simple dialog box should not result in pages of diagrams. For logical hierarchies that become too deep or too wide, you may need to redesign your UI.

Figure 2-1 shows what an e-mail address window containing two child elements and its corresponding logical hierarchy looks like.

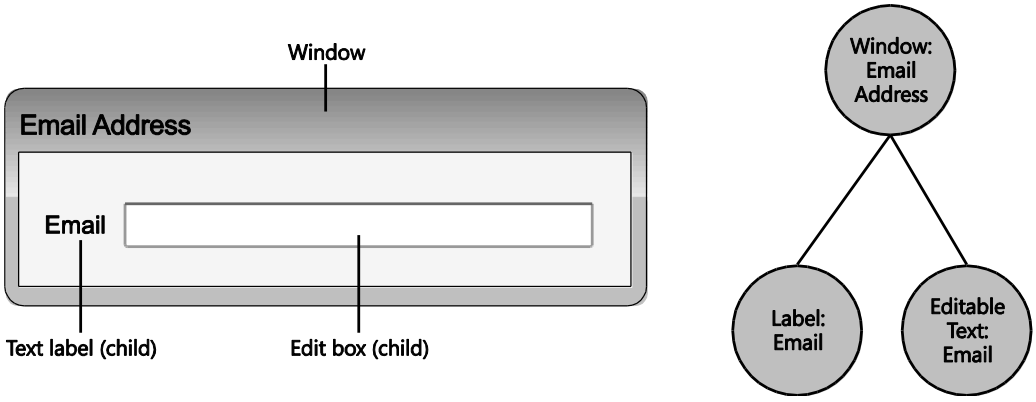

FIGURE 2-1 UIA Provider with two child elements and its corresponding logical hierarchy

When diagrammed, a logical hierarchy will look like a tree, but this "tree-like" structure should not be confused with the UIA Tree. The logical hierarchy is a tool in your specification used to help design the user experience. It is an abstraction of your application's UI and the founda-

tion for accessible software design. Designing a logical hierarchy will also help you understand how to map the control's functionality and features in UIA, which we cover in the next chapter, and it will help to reveal any constraints or hidden costs in advance, as well.

By taking the time to identify and design the logical hierarchy of your UI, you will be on your way to turning over a very usable and accessible product.

Mapping Basics

To create a logical hierarchy, you will examine the layout of your UI to determine how you want your user to navigate through the elements. Then, for each control, you will identify whether they are common or custom controls and map them accordingly. Before we walk through these steps in greater detail, let's go over some basics you should know about elements, controls, element relationships, and navigation when mapping a logical hierarchy.

Elements and Controls

UI elements are the most basic "building blocks" in a logical hierarchy. They are either controls provided by the framework or are exposed as an element with separate functionality by other elements.

Some frameworks have controls that other frameworks do not. If you are using the framework's control as is, you do not need to break down the control any further and map out any child elements that make up that control in your logical hierarchy. The framework already provides a majority of the programmatic access for the control, so the control can be mapped as a single element. For example, because Win32 common controls have a "Menu" control, you would only need to map the Menu control as a single element.

On the other hand, in the case of a developer using HTML, the "Menu" control does not exist. So, the individual elements that make up the control, such as a menu bar, menu items, and pop-up menus, would need to be represented in a logical hierarchy to ensure that programmatic access for these items are implemented.

Naming Elements

As you learned in Chapter 1, "The UI Automation Environment," AT programs and their users depend on the `Name` Property of an element, so be sure to include an accessible name with each element that you map. Consistent naming practices are very important. An accessible name should be consistent with the UI text on-screen, for example.

For images and visual UI elements, the accessible name can sometimes be alternative text, which gives users context about the graphic. For instance, an icon with only an exclamation mark may have a name of "Alert" to tell users what the graphic is about.

Containers

Any element that bounds another object or group of objects is called a "container." For example, a data grid is a container, composed of individual grid items. Those individual grid items may also have elements that contain other elements.

When designing a logical hierarchy, you should only focus on containers that are useful for UI operations and providing context. Avoid including any grouping elements that are purely programmatic or only for visual design. For example, do not include a layout element that only adds redundancy or a graphical element that is hardly named (such as a background image for branding). Without these types of elements, AT clients can more easily filter elements when navigating different views of the UIA Tree.

Element Relationships and Navigation

You should already be familiar with parent/child and sibling relationships. Every element has a relationship, relative to the application window, which contains all UI elements in the application. Elements that share the same parent, such as the application window, are siblings.

The order in which sibling elements appear in the logical hierarchy is particularly important because the exact model will be used by screen readers and other AT to relay to users what they will hear and experience.

Take a look at how the elements in a data entry group box are numbered in Figure 2-2.

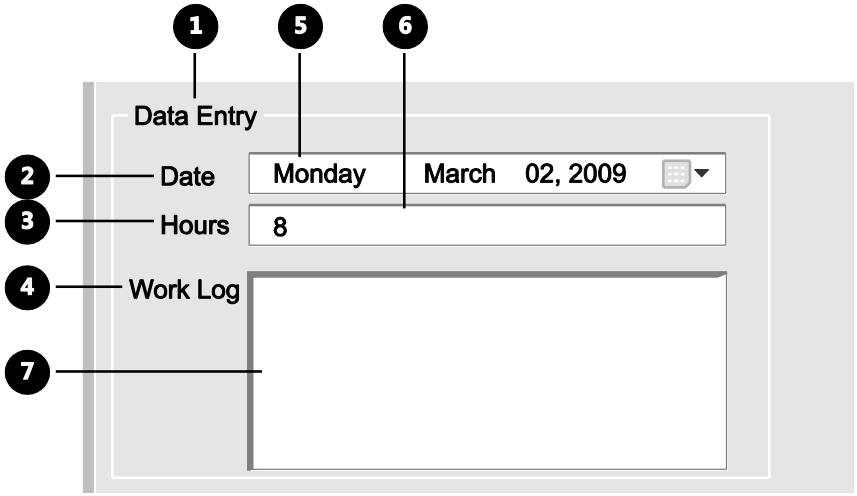

FIGURE 2-2 Elements in a data entry group box using a poor navigational order

If a screen reader were to read and follow the UI structure by the exact order in Figure 2-2, it would read the UI incorrectly, as in Figure 2-3. It may read the UI as follows: "...Data Entry, Date, Hours, Work Log, Work Log date: Monday, March 2, 2009, (blank) nameless editable text, (blank) nameless editable text..." The user would have a very difficult time trying to fill out the crucial pieces of information in their timecard. Because the Date, Hours, and Work Log labels are not read with their corresponding fields, the user may have a hard time entering information for these three things.

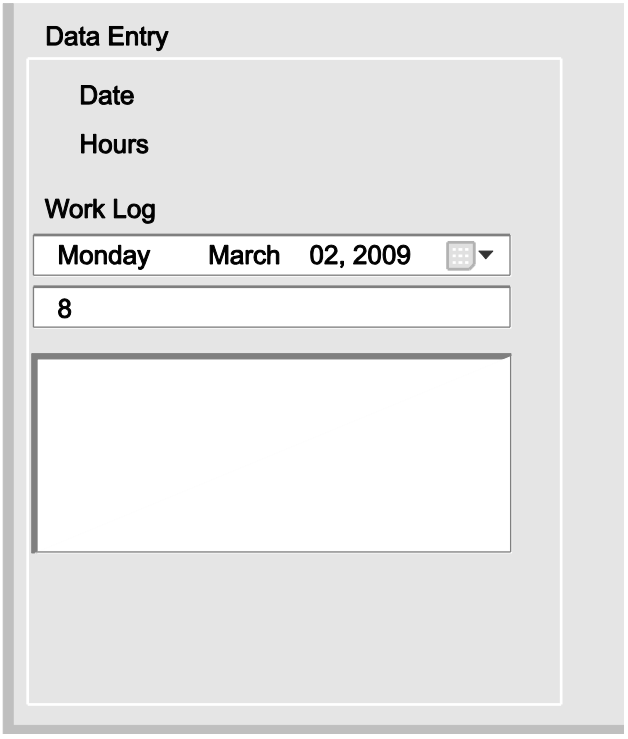

FIGURE 2-3 UI representation of a data entry group box to a screen reader following the poor navigational order of Figure 2-2

Be sure to examine the layout of your UI and the relationships between elements. How would you want your user to read through the interface? What navigational order makes the most sense? What sequence would allow a user to understand the UI most intuitively? Determine what controls relate to each other (for example, a label and its corresponding edit box). Someone who is blind must be able to navigate your UI in a logical and easy way. It is not surprising that accessible UI design shares a lot of best practices and guidelines with usability and UI design guidelines.

Standard Mapping Scheme: Top to Bottom, Left to Right

Although the standard mapping of the logical hierarchy follows a top-to-bottom, left-to-right scheme (a *depth-first search* tree traversal pattern) of the UI, AT clients can interpret the logical hierarchy however they want. That is, the clients can examine or navigate through the elements following a different pattern, such as from the bottom up or right to left. As long as the parent/child and sibling relationships are represented correctly and optimally, the logical hierarchy can be localized to fit the users' needs.

Getting Started

There are four things that you need before you start to design a logical hierarchy:

1. **Format** How you format your logical hierarchy is up to you, but your engineering team should decide how you want it represented before you begin mapping. You can map the logical hierarchy visually using a node-link diagram (as in Figure 2-1) or textually using an outline or table format.

   Mapping in an outline format may look something like the following:

   I. Window: Product Name

   A. Element: Name (top-level child)

   B. Element: Name (top-level child)

   a. Element: Name (second-level child)

   b. Element: Name (second-level child)

   i. Element: Name (third-level child)

   C. Element: Name (top-level child)

   Mapping in a table format may look something like Table 2-1.

TABLE 2-1 Template for Mapping in a Table Format

Window: Product Name

Parent Element	Child Elements
Element: Name (top-level child)	Element: Name (second-level child)
	Element: Name (second-level child) • Element: Name (third-level child)
Element: Name (top-level child)	Element: Name (second-level child)

When mapping with a diagram, use the mapping symbols in Table 2-2 for your logical hierarchy.

TABLE 2-2 Logical Hierarchy Mapping Symbols

Symbol	Represents
Circle O	UI element
Solid line —	Parent/child relationship
Ellipsis ...	More siblings or repeat elements
Asterisk *	Custom control

In addition, you can use color to further differentiate custom controls from standard controls.

2. **UI prototypes** Paper prototypes, computer drawings, UI code mockups, etc. Any prototype will do, just make sure you have enough variations of the prototype to consider different modes of the UI if there are any.

3. **Control libraries of your choice** You will refer to the control library to determine whether a control is provided by the UI framework, as well as to help you correctly identify the control type to add to your logical hierarchy.

4. **UIA Specifications for Control Types, Patterns, and Properties** The technical reference will help you determine whether a custom control can map to a UIA Control Type or other Properties. The specifications can be found at *http://go.microsoft.com/fwlink/?LinkId=150842*. Table 2-3 lists 39 Control Types supported in UIA.

TABLE 2-3 Control Types supported in UI Automation

UI Automation Control Types		
Button	Image	SplitButton
Calendar	List	StatusBar
CheckBox	ListItem	Tab
ComboBox	Menu	TabItem
Custom	MenuBar	Table
DataGrid	MenuItem	Text
DataItem	Pane	Thumb
Document	ProgressBar	TitleBar
Edit	RadioButton	ToolBar
Group	ScrollBar	ToolTip
Header	Separator	Tree
HeaderItem	Slider	TreeItem
Hyperlink	Spinner	Window

How to Do It

The steps in this section should provide you with a quick start on how to design your logical hierarchy. The example that follows provides further discussion.

1. The product window is parent to all the elements contained in it. Map the product window at the top of your logical hierarchy, and label the element using its Control Type and the name you assign it, such as the "Window: Email Address" node in Figure 2-1. If you are using an outline or a table format, this element would be the first item in your outline or a header 1 (see the previous section, "Getting Started").
2. Examine the layout of your UI to determine how you want your user to navigate through the elements in it. Note which elements are grouped together or relate to one another, such as labels and their corresponding fields. Navigation between siblings should be by tab stops and arrow keys for elements within a grouping. As you design your logical hierarchy, you must ensure that the structure reflects the parent/child and sibling relationships of your UI to allow for AT users to easily navigate through it. Prototyping can help with this step.
3. Identify custom controls, whether brand new or ones that have been modified with a different functionality on an existing framework control. For instance, the Win32 list view control does not support a check box, but if you modified the control so that it does have a check box, you would identify the control as a custom control.
4. For each programmatically significant element (that is, an element necessary for UI operations or for giving ATs context), map the control type and name the element (and child elements) as follows:
    - **Standard** Map the node as a single element if the control is based on standard control customizations. For a standard combo box, for instance, you would not need to map an element for the open and close button or list box in the control because the detailed mapping within the "combo box control" is already implied.
    - **Custom** Map the individual elements that make up that control in the logical hierarchy, if the control is new or customized based on a standard control of the UI framework. If possible, try to find an associated UIA Control Type. Chapter 3, "Designing Your Implementation," touches more on this topic.

    Table 2-4 lists a series of questions that will help you identify elements that should be included in your logical hierarchy.

TABLE 2-4 Questions to identify an element to be mapped in a logical hierarchy

Question	Considerations
Question 1: Does the framework provide the control?	If yes, map the control as a single element in your logical hierarchy, and move on to the next control in your UI. If no, proceed to Question 2.
Question 2: Does the control map to a Control Type in UIA?	Each UIA Control Type has required and optional Properties and Control Patterns. If it is difficult to map an element to a UIA Control Type, identify the types of UI functions it exhibits, and map the functionalities to the appropriate UIA Control Patterns and Properties. While UIA allows for a "Custom" Control Type, a control can be identified by the levels (different elements) or enhancements (different functionalities) used for the existing Control Type. For example, the RangeValue Control Pattern could be an enhancement in a combo box Control Type used to support loading status information. If the element does not meet any of the specifications for a UIA Control Type, consider splitting the element into sub-elements if the control is a mix of multiple Control Types, and return to Question 1 for each sub-element.
Question 3: Can you interact with parts of the control with the keyboard alone?	Every action that is provided by the mouse must also be provided by the keyboard. Be careful not to confuse selection for focus. Mouse "hot tracking" is also sometimes confused as selection or focus. If keyboard-only navigation becomes too difficult, consider an alternate way of grouping the elements in your UI or redesigning the hierarchy.
Question 4: Can the control's functionality be defined completely by Control Patterns and Properties?	You may have already answered this question in Question 2 if the element maps to a UIA Control Type. Make sure all possible Patterns and Properties are mapped based on UI scenarios and functions, and reconfirm that you're not violating rules and requirements for each Control Type specification. If the answer to this question is no, identify missing features and functions. Consider using a different Control Type or logical structure. Breaking down the control into smaller elements can sometimes help avoid missing features or functions.

Example: Employee Timecard

To demonstrate how to design one logical hierarchy, we will use an employee timecard application built on a Win32 framework, as an example. Figure 2-4 shows what the timecard looks like.

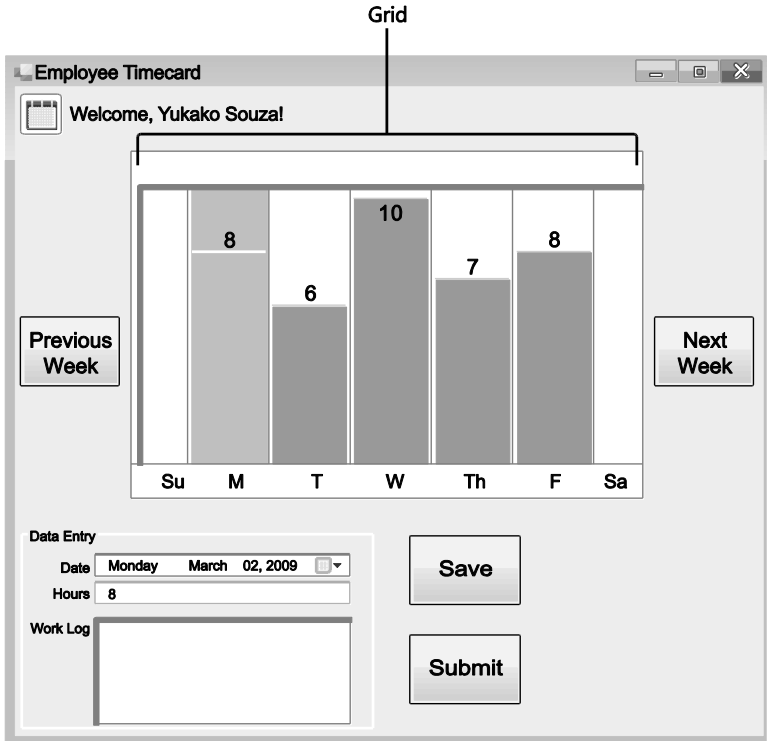

FIGURE 2-4 Employee timecard built on a Win32 framework

In the timecard, employees can:

- Click a date on the grid to see their hours or work log notes populate in the Data Entry fields.
- Use the arrow keys on the keyboard to navigate through the days in the grid.
- Enter their hours in the Hours field.
- Enter notes about their work in the Work Log field.
- Click the Previous Week button to see the previous week, and the Next Week button for the next week.
    - At the start of the fiscal year, the Previous Week button will not be available because the system archives the previous year, and employees will no longer have access to those weeks.
    - If employees are on the current week, the Next Week button will not be available because they cannot log their hours or work for future weeks.

- Save an entry without submitting.
- Submit a week for payroll review.

Except for the grid, all controls in the timecard are standard Win32 controls.

Navigational Order

Looking at the timecard, we see that there are two visual containers in the UI: the grid, made up of columns for each day of the week, and the Data Entry box, which contains the Date, Hours, and Work Log fields. Because these items are grouped together, and the fields within the container are closely related, we must ensure that the order in which we map these items must follow one another logically. Following a general top-to-bottom, left-to-right scheme, Figure 2-5 shows the navigational order in which we will map the logical hierarchy.

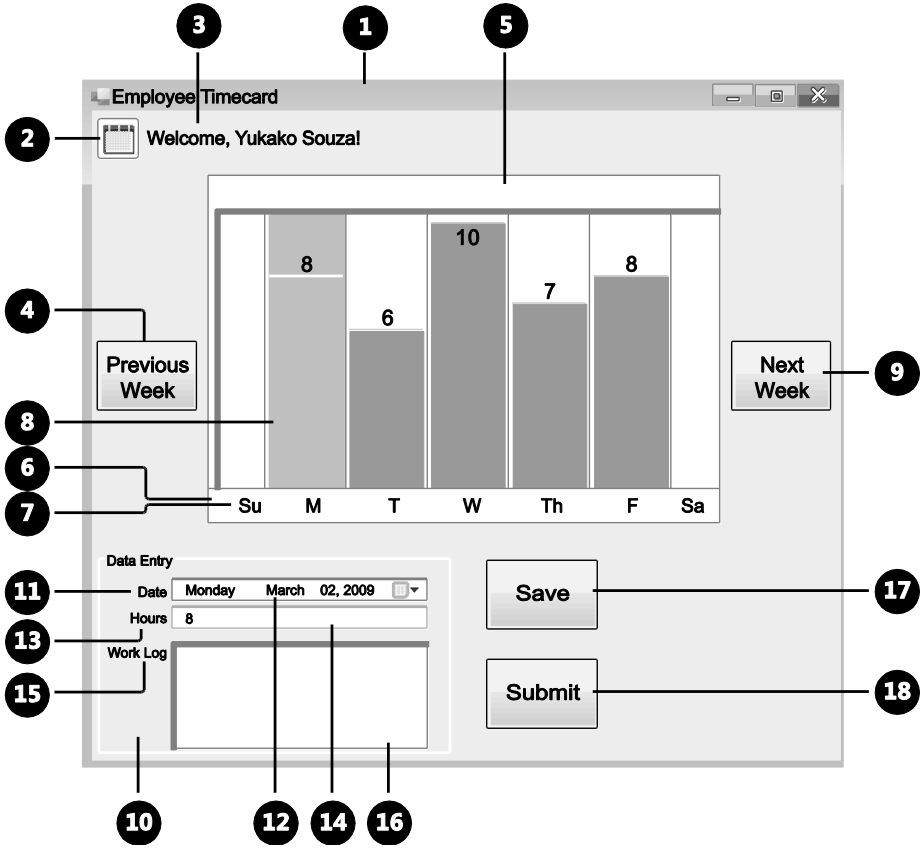

FIGURE 2-5 Navigational order for mapping the timecard's logical hierarchy

Mapping the First Element: Window

Now, we can start mapping. The window element containing the timecard application is mapped at the top of the logical hierarchy and named "Window: Timecard."

Standard Controls: First Three, Top-Level Children

The next three controls are the calendar image, the "Welcome, Yukako Souza!" label next to it, and the Previous Week push button. Looking at the Win32 control library, we see that the framework provides controls for these items, so they are standard controls and can be mapped as single elements on our logical hierarchy.

Below the window element, we plot the first three, top-level child elements from left to right according to their numerical navigational order (Figure 2-6). To indicate the parent-child relationships to the window, we draw lines from the child elements to the parent element.

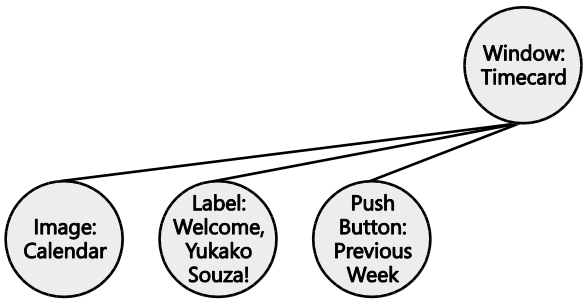

FIGURE 2-6 First three, top-level child elements of the employee timecard

Custom Control: Grid

The next control that we need to map is the grid. Looking at the Win32 control library, we see that there is not a control that captures all of the functionality of our timecard grid. It is, therefore, a custom control, which means we must break down the grid control into elements that make up the UI fragment for that control (as it might be seen in the UIA Tree). But which elements do we map? Using the questions in Table 2-4, we can identify these elements:

- Question 1: Does the framework provide the control? No. We move onto Question 2.
- Question 2: Does the control map to a Control Type in UIA? Yes. Looking at the UIA Specification for some sort of grid control, we see that our timecard grid supports the requirements for the DataGrid control. We also see that the required tree structure includes any headers and data items. In our timecard, the header is the row of labels running underneath the columns (Su, M, T, W, Th, F, and Sa), and the columns are the data items.

Our logical hierarchy now looks like Figure 2-7. Note that because there are several grid item and header elements, we mark those nodes with an ellipsis to indicate that there is more than one element for that Control Type (see Table 2-1 for mapping symbols).

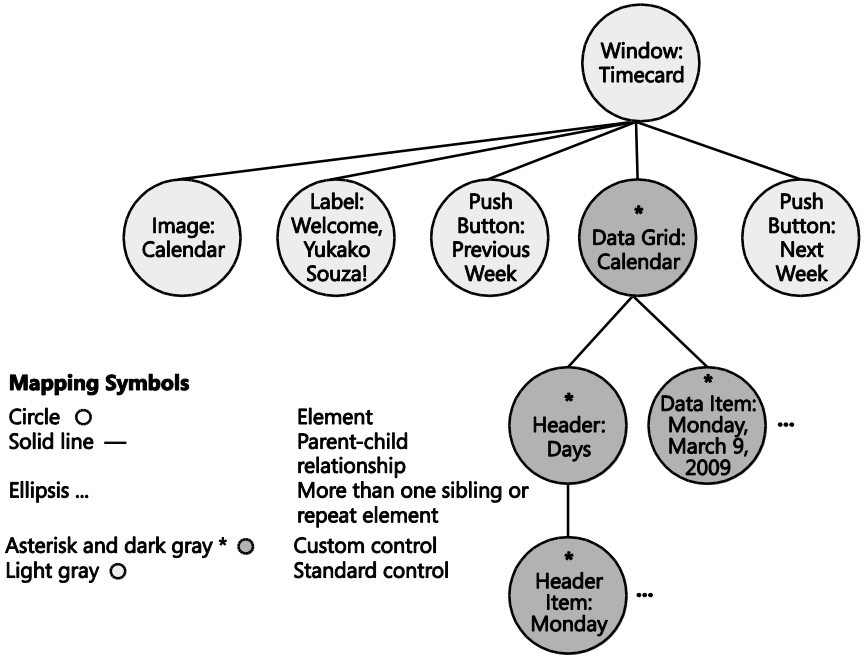

FIGURE 2-7 Grid element added to the employee timecard's logical hierarchy

Determining the elements to map for the grid may have seemed fairly straightforward, but sometimes it is not that easy. Let's say that we weren't sure about the grid's functionality. Instead of mapping the grid to the DataGrid control, we make the mistake of identifying the columns as push buttons, because when we click them, they interact very much like push buttons. Let's see how we might have worked through this process.

- Question 1: Does the framework provide the control? No. We move onto Question 2.
- Question 2: Does the control map to a Control Type in UIA? Yes. When we click one of the columns, the interaction is very much like clicking a push button. For now, let's say that the columns are all push buttons, which can be mapped to the `Button` Control Type in UIA.

- Does the element meet the UIA Control Type Specification requirements completely? No. We see that one of the Properties for the `Button` control is that buttons are self-labeled by their contents, as with an "OK" or "Save" button. In our timecard, our "buttons" (the clickable columns) are not labeled as such, but instead have labels with the days of the week running underneath them. We could argue that the number of hours that appear on the columns are labels for the "buttons," but the value ("8" for 8 hours, for instance) does not accurately describe the column nor is it constant (some days may not even have any hours entered, for instance). We must, therefore, start the process over again.

Taking a step back and looking at the grid as a whole, we see that that the grid is (and by definition, should be) made up of rows and columns. Each day is a clickable column, and the group of labels that runs in a row underneath the columns is actually a header for the days of the week. Looking at the UIA requirements for the DataGrid Control Type, we see that a data grid must have data items within that control. At this point, we can deduce that the clickable columns are data items (and not buttons). To verify, we check the requirements for the Control Type in the UIA Specification and confirm that the clickable columns meet the conditions for the DataItem Control Type. The columns are, in fact, data items, elements that we can map in a logical hierarchy. A close examination of the UIA Specification can save you time and answer a lot of design questions because the structures for controls are clearly defined.

Container: Data Entry Group Box

The remaining controls are all Win32 common controls and can be mapped as single, standard elements. As mentioned earlier, however, the Data Entry group box is a visual container in the UI for the Date, Hours, and Work Log fields and their corresponding labels. We must be sure to reflect these parent/child relationships in the logical hierarchy. Figure 2-8 illustrates what the completed logical hierarchy looks like for the timecard application.

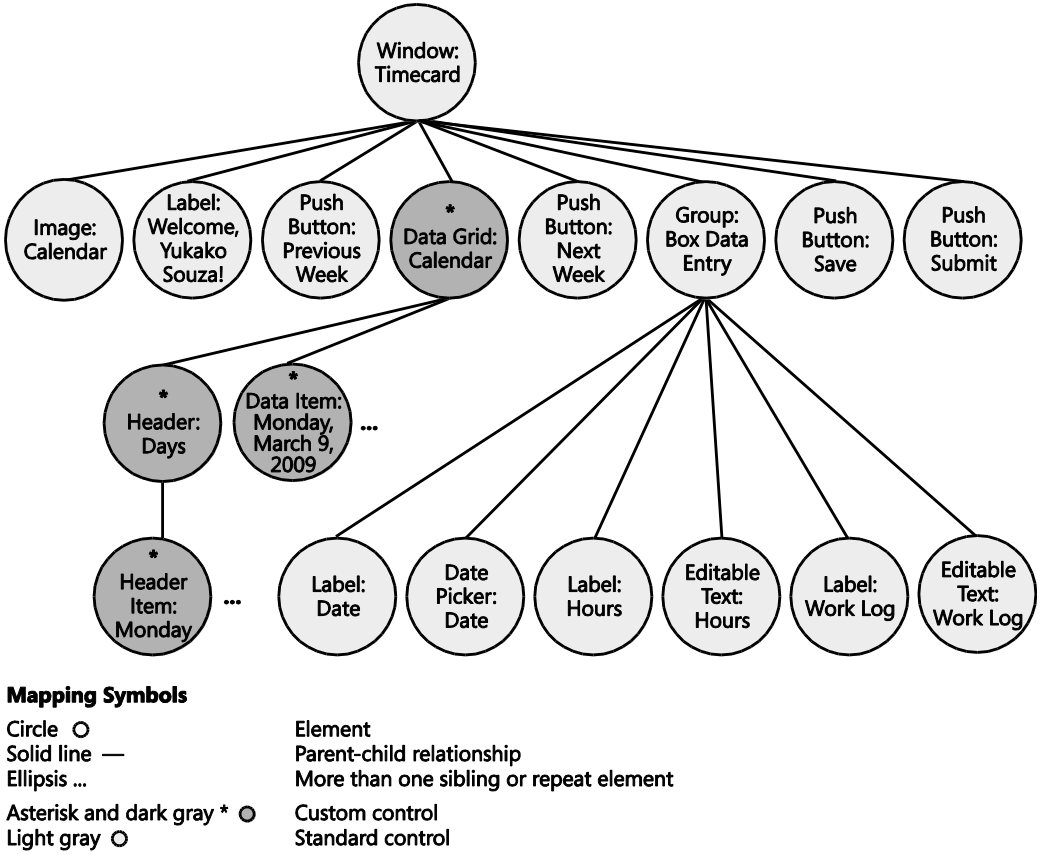

FIGURE 2-8 Completed logical hierarchy for employee timecard

Using the Logical Hierarchy for Planning Accessibility Settings

After plotting out the elements of your UI, the logical hierarchy can be used to assist with planning other accessibility settings, such as keyboard navigation and graphics.

Keyboard Navigation

Because your controls are already laid out in a logical hierarchy, it is easy to design your keyboard navigation. Controls that the user can interact with, such as buttons, links, or list boxes, should receive keyboard focus and may need to be part of a tab-stop loop in the keyboard navigation. Users should be able to move between controls using the TAB key and SHIFT+TAB. For grouped elements, you may need to ensure sub-navigation routines using arrow keys within two dimensional grids, or even CTRL+TAB to move between the grouped elements. If your UI supports multiple-selection, you may need to support SHIFT+RIGHT ARROW and SHIFT+LEFT ARROW key combinations.

Go further: For more information on designing keyboard navigation and UI design, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Graphics: Decorative vs. Contextual

Your logical hierarchy can also help you identify decorative elements from contextual elements in your UI and the order in which they should be read by an AT program. Because the logical hierarchy is a rather primitive representation of your UI design, you should not have very many decorative UI elements in the hierarchy, because the user does not typically need to interact with graphics. Only graphics that play a crucial role in the UI's messaging should be included, such as notification or information icons, and the order of the information about the graphical information should not interfere with other important information in the UI. For instance, information about a background graphic in the UI should not appear in the logical hierarchy where it would interfere with critical information for the user. Identifying which graphics are decorative and contextual and determining where they should appear in the logical hierarchy will help with filtering any trivial elements in the object model.

Go further: UIA can filter out non-control or non-content elements by allocating elements with both the IsContentElement and IsControlElement Properties set to FALSE. For more information about how to choose and set values for those Properties, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Complex User Interfaces

The logical hierarchy for the employee timecard that we just designed was fairly simple, but user interfaces are becoming more complex with richer functionality. As you create logical hierarchies for your UI, keep these principles in mind:

- Create logical hierarchies for all UIs that you design to ensure "seamless accessibility" for your users. Any new child window that your application creates, such as pop-up windows, should have its own logical hierarchy and accessible implementation.

- Take advantage of UI framework–provided controls and components. Just as you want to use built-in controls, using standard controls enables you to get some programmatic access "for free." Again, using these components may require you to adhere to certain accessibility guidelines and restrictions on the controls, but those have a much lower cost than a completely native UIA solution. For example, Windows Common Controls provides a list view control that can easily be implemented into your design, but the accessibility support for an irregular customization of a list view control may be extremely expensive when what you really wanted was an "engineering shortcut."

- Keep the UI as intuitive as possible. As mentioned, accessibility shares best practices and requirements with many usability and UI design guidelines. Always remember that the more complex and unique your user interface, the more work you will have to do to make it accessible. If you can accomplish your requirements in a usable, accessible, and aesthetically pleasing manner using framework controls and components, then your costs for implementation and testing will be much less than when you have to use custom controls.

Go further: For other components provided by Windows, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Designing Element Functionality

Elements are the building blocks of your UI's logical hierarchy. By mapping out the programmatic access for your application in a logical hierarchy, you help to ensure that client programs, such as AT and automation tools, can navigate the UI and that users can confidently use your product. In the next chapter, we discuss how to determine the implementation of your controls, with particular focus on the design of custom controls in your logical hierarchy.