Chapter 3

Designing Your Implementation

After you have finished designing your logical hierarchy, you should know which controls in your product are provided by the UI framework and which are not. Designing the implementation of your controls depends upon this distinction:

- For controls provided by the framework, you must adhere to the UI framework's guidelines to make them accessible. For example, if you are using the Windows Presentation Foundation (WPF) framework, you would adhere to WPF's guidelines for accessibility.
- For custom controls not provided by the UI framework, you must implement a native UI Automation (UIA) solution. You have already mapped these custom controls to individual elements in the logical hierarchy, so now you must design the native UIA solution for each of these elements.

The key to designing a native solution for programmatic access is to fully expose the element's functionality so that a user of assistive technology (AT) can use the control. There are two different processes for designing the implementation of a native solution:

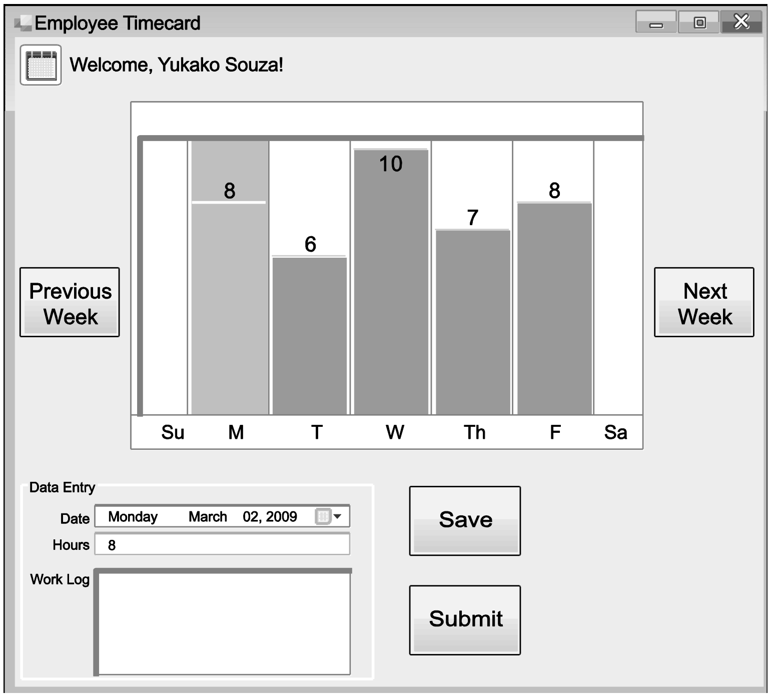
A. **Control maps to a UIA Control Type.** If your custom control can map directly to a UIA Control Type, you must design the control's functionality according to the UIA Control Type Specification, including any additional requirements for other Patterns and Properties that the control may exhibit. Unless it is prohibited, a Control Type can support additional Patterns and Properties than what is required or suggested by the UIA Specification.

B. **Control does *not* map to a UIA Control Type.** In the case where your custom control does *not* map to a UIA Control Type, then you must determine the control's functionality and design the control around the Control Patterns and Properties using the requirements of the UIA Specification. It is worth noting again that you should avoid creating new custom controls as much as possible because the cost for development, documentation, and help on how to interact with the control is significant, and ATs may not know how to interact with the control.

In this chapter, we talk about both of these design processes, focusing on controls that do map directly to a UIA Control Type. We also touch on the UIA Methods and Events that are needed to implement your controls and point you to resources for actually implementing them.

Product Example Continued: Employee Timecard

In the last chapter, we used an employee timecard, built on a Win32 framework (Figure 3-1), to design a logical hierarchy. We continue to use the timecard in this chapter to demonstrate how to design the implementation of custom controls.

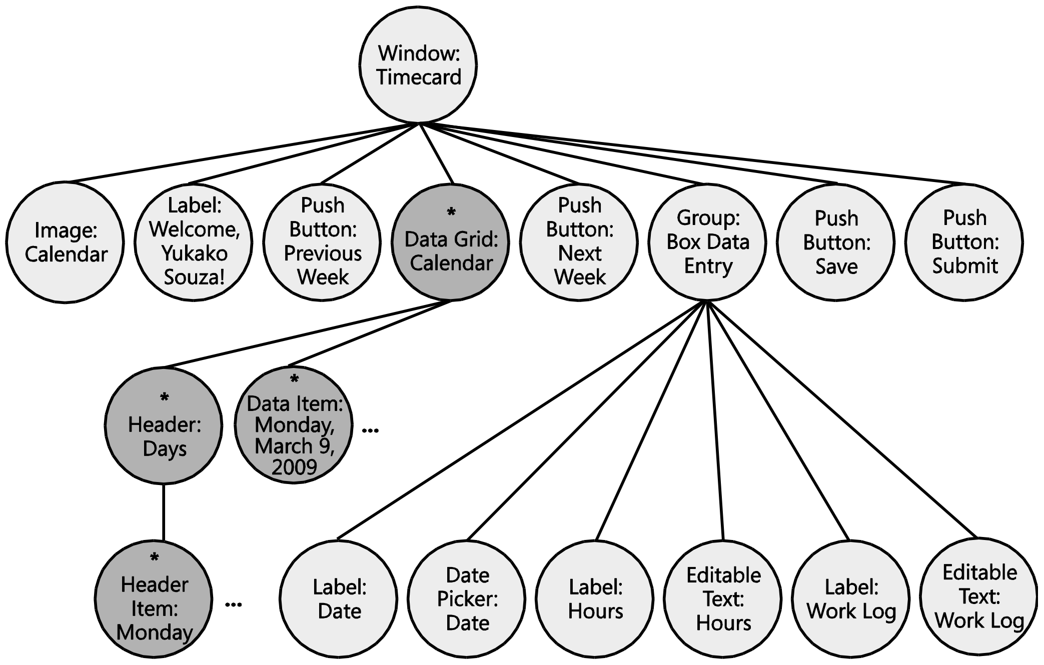

Welcome, Yukako Souza!

Date: Monday March 02, 2009

Hours: 8

FIGURE 3-1 Product example: employee timecard built on a Win32 framework

As you may recall, all the elements in the timecard, except for the grid, were Win32 common controls. By mapping out a logical hierarchy for our timecard (Figure 3-2), we can see where custom accessibility support is needed. Because Win32 does not provide a “Grid” control, we needed to map out the individual elements that make up that the control, so that the control will expose correctly to AT.



Mapping Symbols

Circle ○	Element
Solid line —	Parent-child relationship
Ellipsis ...	More than one sibling or repeat element
Asterisk and dark gray * ●	Custom control
Light gray ○	Standard control

FIGURE 3-2 Logical hierarchy for the employee timecard

Prep Work: Creating the Implementation Table

By now, you should have an understanding of what Control Types, Control Patterns, and Properties are. Before we proceed, let's briefly recap these terms:

- **Control Type** A pre-defined set of patterns, properties, and conditions used to define a control's basic appearance and functionality.
- **Control Pattern** Defines the control's actions or behaviors.
- **Properties** Provides specific information about the UI element or the Control Patterns supported.

When you design a native solution for a custom control in UIA, you are essentially creating an engineering "recipe" using the UIA Specification for UIA Control Types, Control Patterns, Properties, and Events. These “ingredients” together will be used to implement an accessible custom control.

Before we proceed with designing our controls, let’s do some prep work. We will create an implementation table for the primary components of the UI:

1. Create columns with the following headers:
   - **Control** For the elements identified as custom in your logical hierarchy.
   - **Control Type** For the UIA Control Type of the element.
   - **Control Patterns** For the required patterns necessary to implement the accessibility of the control.
   - **Properties** For the required automation element and control pattern properties necessary to implement the accessibility of a UI element feature.
2. Using your logical hierarchy as a reference, list each custom element in the Control column. You can omit duplicate elements, such as list items or data items that share the same characteristics with its peers. For example, the employee timecard has seven unique controls for “Grid Item: Days,” but the design for each instance will be the same (except for unique Properties such as the Automation Id).
3. In the Control Type column, list the UIA Control Type that the element maps to. Again, you should have this information as a result of mapping out the logical hierarchy for your product.

Table 3-1 illustrates what the implementation table looks like for the employee timecard so far.

TABLE 3-1 Employee Timecard Custom Controls

Control	Control Type	Control Patterns	Properties	
			Automation Element Properties	**Control Pattern Properties**
Data Grid: Calendar	Data Grid			
Grid Item: Days	Grid Item			
Header: Days	Header			
Header Items: Days of Week	Header Item			

Process A: Control Maps to a UIA Control Type

Designing the implementation for custom controls that map to a UIA Control Type is a two-part process. You will:

1. Gather all the UIA Specification requirements for the UIA Control Type and list them in your implementation table.
2. List any additional Patterns or Properties for the controls if they exhibit any additional functionality, but make sure those additional Patterns or Properties do not contradict with the UIA Specification.

All of the controls map to UIA Control Types in our employee timecard application, so we proceed with Process A.

Step 1: Gathering Required Control Patterns

The first control in our table is the calendar grid, which maps to the `DataGrid` Control Type. The UIA Specification provides a table of required Patterns supported by the Data Grid Control Type (Table 3-2). We must go through each of these Patterns to verify which apply to our specific custom control.

TABLE 3-2 Required UI Automation Control Patterns for the DataGrid Control Type from the UIA Specification

Control Pattern	Support	Notes
Grid Pattern	Yes	The data grid control itself always supports the Grid Control Pattern because the items that it contains have metadata that is laid out in a grid.
Scroll Pattern	Depends	The ability to scroll the data grid depends on content and whether scroll bars are present.
Selection Pattern	Depends	The ability to select the data grid depends on content.
Table Pattern	Depends	A data grid control that has a header should support the Table Control Pattern.

Among the Patterns listed, only the Grid Pattern must always be supported by controls using the `DataGrid` Control Type. The Scroll Pattern, Selection Pattern, and Table Pattern, however, are dependent upon the specific data grid. Because the calendar grid in our timecard application does not scroll, the Scroll Pattern does not apply. The user can, however, select items in our grid, so the Selection Pattern also applies. Finally, our grid does support headers (which run underneath each column), so it supports the Table Pattern, as well. In our implementation table, we would, thus, list the Grid, Selection, and Table Patterns under the Control Patterns column for our timecard grid (Table 3-3).

TABLE 3-3 Required Control Patterns for the employee timecard's calendar grid custom control

Control	Control Type	Control Patterns	Properties	
			Automation Element Properties	Control Pattern Properties
Grid: Calendar	DataGrid	Grid Selection Table		

Step 2: Gathering Required Control Type Properties

The next step is to fill out our columns for the two types of Control Properties:

1. Automation Element Properties
2. Control Pattern Properties

Go further: For UI Automation Element and Control Pattern Properties, go to http://go.microsoft.com/fwlink/?LinkId=150842.

2a. Required Automation Element Properties

The Automation Element Properties listed for each Control Type is a subset of all the Automation Elements available that are likely to describe the element. The `AutomationId` and `Name` Properties appear on all Property lists for UIA Control Types. For the `DataGrid` Control Type, the UIA Specification lists Automation Element Properties whose value or definition is particularly relevant to `DataGrid` controls (Table 3-4).

TABLE 3-4 UI Automation Properties for the DataGrid Control Type from the UIA Specification

Property	Value	Notes
`AutomationId`	See notes	The value of this Property needs to be unique across all controls in an application.
`BoundingRectangle`	See notes	The outermost rectangle that contains the whole control.
`ClickablePoint`	See notes	Supported if there is a bounding rectangle. If not every point within the bounding rectangle is clickable, and you perform specialized hit testing, then override and provide a clickable point.
`ControlType`	DataGrid	This value is the same for all UI frameworks.

Property	Value	Notes
`IsContentElement`	True	The value of this Property must always be True. This means that the data grid control must always be in the content view of the UI Automation tree.
`IsControlElement`	True	The value of this Property must always be True. This means that the data grid control must always be in the control view of the UI Automation Tree.
`IsKeyboardFocusable`	See notes	If the control can receive keyboard focus, it must support this Property.
`LabeledBy`	See notes	If there is a static text label, then this Property must expose a reference to that control.
`LocalizedControlType`	See notes	Localized string corresponding to the DataGrid Control Type. The default value is "data grid" for en-US or English (United States).
`Name`	See notes	The data grid control typically gets the value for its Name Property from a static text label. If there is not a static text label, an application developer must assign a value for the Name Property. The value of the Name Property must never be the textual contents of the edit control.

For all 10 Properties, we can apply values specific to the timecard's calendar grid. For the `AutomationId`, `BoundingRectangle`, `ClickablePoint`, `IsKeyboardFocusable`, `LabeledBy`, `Name`, and `LocalizableControlType` Properties, which have no specified value, we must refer to the UIA Specification to find the data type for the values needed for the Property. For each of these variable Properties, we specify the Property values for the timecard in Table 3-5. Note that the `ClickablePoint` Property is omitted because it is irrelevant for the timecard's grid.

TABLE 3-5 Variable Automation Element Property values assigned for custom calendar grid control

Automation Element Property	Value	Data Type	Notes
`AutomationId`	TimecardGrid	VT_BSTR	The value for the `AutomationId` should be unique among siblings.
`BoundingRectangle`	Coordinates of table onscreen	VT_R8|VT_ARRAY	The value of the rectangle is expressed in physical screen coordinates.
`IsKeyboardFocusable`	False	VT_BOOL	The grid itself cannot receive keyboard focus; only the grid items can.
`LabeledBy`	Null	VT_UNKNOWN	Null because there is no text label for the grid.

Automation Element Property	Value	Data Type	Notes
`Name`	"Calendar"	VT_BSTR	Typically, the value for the `Name` Property should match the label text on screen. Because there is no on-screen label, "Calendar" is assigned. In combination with the `LocalizedControlType` Property, the control may read as "Calendar timecard grid."
`LocalizedControlType`	"timecard grid"	VT_STR	`LocalizedControlType` can be modified to be more understand able to the user. For English, it is suggested that the string for the `LocalizedControlType` Property be typed in small caps because it will be used in-line with the Name Property.

With the required Automation Element Property values now defined, you can fill out the Automation Element Properties column for the calendar grid. Table 3-6 shows what our table looks like so far.

TABLE 3-6 Implementation table with the required Automation Element Properties and their values for the employee timecard's calendar grid custom control

Control	Control Type	Control Patterns	Properties	
			Automation Element Properties	Control Pattern Properties
Grid: Calendar	DataGrid	Grid Selection Table	• `AutomationId`: TableHeader • `BoundingRectangle`: Coordinates of table onscreen • `ControlType`: DataGrid • `IsContentElement`: True • `IsControlElement`: True • `IsKeyboardFocusable`: False • `LabeledBy`: Null • `LocalizedControlType:` "timecard grid" • `Name`: "Calendar"	

Go further: For data types and properties, go to http://go.microsoft.com/fwlink/?LinkId=150842.

2b. Required Control Pattern Properties

Each Control Pattern in UIA has Properties of their own that we need to implement. Using the UIA Specification again, we can see what Properties are required for each Control Pattern and assign a value for each Pattern Property. Table 3-7 lists the Property name, value assigned, and notes about the Property for each Control Pattern.

TABLE 3-7 Control Pattern Property names and values for the timecard's calendar grid

Control Pattern	Property Name (Data Type)	Value	Notes
Grid Pattern	`ColumnCount` (VT_I4)	7	The total number of columns in a grid. The control has seven columns, one column for each day.
	`RowCount` (VT_I4)	1	The total number of rows in a grid. The control has one row of columns.
Selection Pattern	`CanSelectMultiple` (VT_BOOL)	False	A value that specifies whether the container allows more than one child element to be selected concurrently. The user can only select one column at a time, so the value is false.
	`IsSelectionRequired` (VT_BOOL)	False	A value that specifies whether the container requires at least one child item to be selected. Employees are not required to select a column when viewing their timecard, so the value is false.
Table Pattern	`RowOrColumnMajor` (VT_I4)	Column	The primary direction of traversal for the table. Column is chosen for the timecard because users would generally read the control by date, which is in a column.

Now that we have determined what our Property values should be for each of the calendar grid's required UIA Control Patterns, we can fill out the Control Pattern Properties column as shown in Table 3-8.

TABLE 3-8 Implementation table with the required Control Pattern Properties and their values for the employee timecard's calendar grid custom control

Control	Control Type	Control Patterns	Properties	
			Automation Element Properties	**Control Pattern Properties**
Grid: Calendar	DataGrid	Grid Selection Table	• `AutomationID`: TableHeader • `BoundingRectangle`: Coordinates of table onscreen • `ControlType`: DataGrid • `IsContentElement`: True • `IsControlElement`: True • `IsKeyboardFocusable`: False • `LabeledBy`: Null • `LocalizedControlType`: "timecard grid" • `Name`: Calendar	Grid Pattern • `ColumnCount`: 7 • `RowCount`: 1 Selection Pattern • `CanSelectMultiple`: False • `IsSelectionRequired`: False Table Pattern • `RowOrColumnMajor`: Column

Step 3: Gathering Requirements for Additional Control Functionality

Now that we have finished listing in our implementation table all the Control Patterns and Properties required by the UIA Specification for a `DataGrid` control, we need to list any additional Control Patterns and Properties that apply specifically to our control.

The question now is "Does my control exhibit additional functionality, aside from the required Control Patterns?" If the answer is yes, then determine what additional UIA Patterns or Properties the control maps to in UIA. If you absolutely cannot find a Control Pattern or Property that exhibits the additional functionality of your control, then you must create custom Control Patterns and Properties to describe your control, or its functionality, and include those in your implementation table. Be aware, however, that your custom specifications are only useful if UIA Clients can share and adopt your specifications. Refer to the UIA Community Promise Specification and resources from the Accessibility Interoperability Alliance (AIA) for best practices and guidance on maximizing usability.

In the case of our timecard's calendar grid, it does exhibit some additional functionality. When the user clicks one of the days in the grid, the Data Entry fields populate with any information that has been previously entered for that day. The grid affects another part of the application, the fields in the Data Entry group box. Because our grid exhibits additional functionality, we must, then, identify and map this functionality to a UIA Control Pattern or Property and list the requirements for that Pattern or Property in our implementation table. Looking at the UIA Specification, we see that the `ControllerFor` Property best describes this other functionality (Table 3-9).

TABLE 3-9 Description of the `ControllerFor` Property from the UIA Specification

Property Name (Data Type)	Description
`ControllerFor` `(VT_UNKNOWN|VT_ARRAY)`	An array of elements that are manipulated by the Automation Element that supports this Property. `ControllerFor` is used when an Automation Element affects one or more segments of the application UI or the desktop; otherwise, it is hard to associate the impact of the control operation with UI elements.

Other than the `ControllerFor` Property, our calendar grid does not appear to exhibit any additional functionality. We will go ahead and add these Properties to our table (Table 3-10).

TABLE 3-10 Completed implementation table for calendar grid custom control

Control	Control Type	Control Patterns	Properties	
			Automation Element Properties	Control Pattern Properties
Grid: Calendar	DataGrid	Grid Selection Table	• `AutomationID`: TableHeader • `BoundingRectangle`: Coordinates of table onscreen • `ControlType`: DataGrid • `IsContentElement`: True • `IsControlElement`: True • `IsKeyboardFocusable`: False • `LabeledBy`: Null • `LocalizedControlType`: "data grid" • `Name`: Calendar • `ControllerFor`: Date Picker, Hours Edit Box, and Work Log Edit Box (This Property can have multiple things.)	Grid Pattern • `ColumnCount`: 7 • `RowCount`: 1 Selection Pattern • `CanSelectMultiple`: False • `IsSelectionRequired`: False Table Pattern • `RowOrColumnMajor`: Column

We have now finished designing the implementation solution for our first custom control element in UIA. Before moving to the next element, it's a good idea to check the UIA Specification's list of Properties to make sure that you have listed all the requirements for your control's functionality. As mentioned, all of our custom controls in the example can map to a UIA Control Type, so we use the same process as the first control (Process A) for each of the remaining elements and fill out the rest of our implementation table (Table 3-11).

TABLE 3-11 Completed implementation table for employee timecard custom controls

Control	Control Type	Control Patterns	Properties	
			Automation Element Properties	**Control Pattern Properties**
Grid: Calendar	DataGrid	Grid Selection Table	• `AutomationID`: TableHeader • `BoundingRectangle`: Coordinates of table onscreen • `ControlType`: DataGrid • `IsContentElement`: True • `IsControlElement`: True • `IsKeyboardFocusable`: False • `LabeledBy`: Null • `LocalizedControlType`: "data grid" • `Name`: Calendar • `ControllerFor`: Date Picker, Hours Edit Box, and Work Log Edit Box (This Property can have multiple things)	Grid Pattern • `ColumnCount`: 7 • `RowCount`: 1 Selection Pattern • `CanSelectMultiple`: False • `IsSelectionRequired`: False Table Pattern • `RowOrColumnMajor`: Column

Control	Control Type	Control Patterns	Properties	
			Automation Element Properties	**Control Pattern Properties**
Grid Item: Days	Data Item	Grid Item Selection Item Table Item	• `AutomationId`: "TC#" (# is replaced by the number of the column from 1 through 7, where "TC1" would be Sunday) • `BoundingRectangle`: Coordinates of grid item onscreen • `ClickablePoint`: any point on screen clicked to select or focus the grid item reliably. • `ControlType`: GridItem • `IsContentElement`: True • `IsControlElement`: True • `IsKeyboardFocusable`: True • `HasKeyboardFocus`: True if the grid item is focused, false otherwise • `ItemStatus`: "data entered" if the grid data is entered, "empty" otherwise • `LabeledBy`: Null • `LocalizedControlType`: "timecard" • `Name`: date of the grid (e.g., "Mon, March 02, 2009")	Grid Item Pattern • `Column`: 1 through 7 • `ColumnSpan`: 1 • `ContainingGrid`: Parent Control • `Row`: 1 • `RowSpan`: 1 Selection Item Pattern • `IsSelected`: True if the grid item is selected, false otherwise • `SelectionContainer`: Parent table/grid control (No Properties for Table Item Pattern)
Header: Days	Header	None	• `AutomationId`: "Header" • `BoundingRectangle`: Coordinates of grid item onscreen • `ControlType`: Header • `IsContentElement`: False • `IsControlElement`: True • `IsKeyboardFocusable`: False • `Labeled By`: Null • `LocalizedControlType`: "header" • `Orientation`: Horizontal • `Name`: "" (Nameless because there is no other header in this control)	

Control	Control Type	Control Patterns	Properties	
			Automation Element Properties	**Control Pattern Properties**
Header Items: Days of Week	Header Item	None	• `AutomationId`: "H#" (# is replaced by the numer from 1 through 7 where H1 is for Sunday) • `BoundingRectangle`: coordinate of header item on screen • `ClickablePoint`: any point on screen clicked to select or focus the associated column • `ControlType`: HeaderItem • `IsContentElement`: False • `IsControlElement`: True • `IsKeyboardFocusable`: False • `LabeledBy`: Null • `LocalizedControlType`: "header item" • `Name`: label string of the element (e.g., "Su" for Sunday header item)	

Go further: For the UIA Community Promise and best practices and guidance on maximizing usability with interoperable implementations, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Process B: Control Does Not Map to a UIA Control Type

So far, we have walked through designing solutions for custom controls if the controls can map directly to Control Types in UIA. What if your custom control does *not* map to a UIA Control Type? If you find yourself in this situation, then you need to take every step to be absolutely sure that your control cannot be mapped to another Control Type. To avoid unnecessary development, documentation, and help costs associated with custom controls, complete the following steps:

1. Try to identify all Patterns and Properties required to describe them.
2. Look at the UIA Control Type list again to see if there is a Control Type sufficient to map to your control. If there is a Control Type that can be used for your control, fill out the appropriate columns in your implementation table with the control's requirements.

   Note that because UIA allows you to add extra Control Patterns and Properties to an existing Control Type (unless prohibited by the UIA Control Type Specification) without making it into a completely new custom control, it is not necessary to match your custom control exactly to a UIA Control Type. You can also offer a customized description of the element based on the existing Control Type with an alternative `LocalizedControlType` Property value.
3. If there is absolutely no Control Type that can be used for your control, the "Custom" Control Type can be applied. Fill out the appropriate columns in your implementation table with the control's requirements, and fill out the `LocalizedControlType` Property with a string that would make sense to AT users.
4. Document and publish your custom Control Type specifications where it is publicly available, following the process defined by a UIA working group of the AIA, so that the specification of the custom control is clear to the users and AT makers. To facilitate the publishing process, it may also be helpful to ask a member of the AIA to publish your specification.

Methods and Events

After determining your Control Types, Patterns, and Properties, you also need to know what UIA Methods and Events are required. Methods, as you may recall from Chapter 1, provide a way to expose a control's functionality per the UIA Specification. Events in UIA are raised to notify clients, such as screen readers or screen magnifiers, that there is a change to the Automation Element in the UI. Determining these Methods and Events is straightforward and usually only requires checking the corresponding Method and Event specifications for Control Patterns and Properties that your control supports. Table 3-12 lists the Properties and Methods that are required to expose the functionality of the three Control Patterns in the timecard data grid.

TABLE 3-12 Control Properties and Methods for the employee timecard's Control Patterns

Control Pattern	Control Properties	Methods
Grid	`ColumnCount` `RowCount`	`GetItem`
Selection	`CanSelectMultiple` `IsSelectionRequired`	`GetSelection`
Table	`RowOrColumnMajor`	`GetColumnHeaders` `GetRowHeaders`

As you learned in Chapter 1, there are many different UIA Events. The UIA Specification directs you on what Events you must raise for your custom control. Table 3-13 lists all the Events that are supported by the data grid element and whether the Event is applicable to our timecard application.

TABLE 3-13 Data Grid UI Automation Events applicable to the timecard's custom grid control

UI Automation Event	Supported
`AutomationFocusChangedEvent`	Yes
`BoundingRectangleProperty Property-changed Event`	Yes
`IsEnabledProperty Property-changed Event`	Yes
`IsOffscreenProperty Property-changed Event`	Yes
`LayoutInvalidatedEvent`	Not applicable. Timecard does not invalidate the layout.
`StructureChangedEvent`	Yes

UI Automation Event	Supported
`CurrentViewProperty Property-changed Event.`	Not applicable. Timecard does not change its view mode.
`HorizontallyScrollableProperty Property-changed Event`	Not applicable. Timecard does not support scrolling.
`HorizontalScrollPercentProperty Property-changed Event`	Not applicable. Timecard does not support scrolling.
`HorizontalViewSizeProperty Property-changed Event`	Not applicable. Timecard does not support scrolling.
`VerticalScrollPercentProperty Property-changed Event`	Not applicable. Timecard does not support scrolling.
`VerticallyScrollableProperty Property-changed Event`	Not applicable. Timecard does not support scrolling.
`VerticalViewSizeProperty Property-changed Event`	Not applicable. Timecard does not support scrolling.
`InvalidatedEvent`	Yes

Framework-Dependent Decisions

This chapter focused on designing your custom controls to meet the UIA Specification, but the design stage does not stop here. Three areas that are framework-dependent that must be determined (if they have not already been determined) are:

1. Your framework's requirements for providing programmatic access to the controls, whether provided by the framework or custom. While standard controls of the UI framework may support the basics for programmatic access, the flexibility for accessibility can be limited to modifications.
2. Determine how UI elements will handle keyboard focus. Controls that are actionable, such as buttons and links, should receive keyboard focus. For Win32 common controls, use the control styles in the resource file, and handle the system focus as needed.
3. Ensure that your UI adheres to other accessibility requirements discussed in the introduction of this book, such as high contrast, high dpi, and other system settings.

Once you have addressed these three areas, you are ready to take your designs into the implementation stage.

Go further: For more information on adhering to accessibility requirements other than programmatic access, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Implementing Your Native UIA Solution

Your next challenge is determining how to implement the native solutions you have designed over the last two chapters. How does your design actually map out to its implementation? How do you take the requirements in your implementation table and actually use the UIA framework to implement it? Because implementation is framework-dependent, this book does not provide specific implementation details, but depending on the complexity of your control, you do need to implement one or more of the UIA interfaces. These interfaces allow you to implement the Control Patterns, Properties, Methods, and Events that you specified in your implementation table.

Go further: For more information on how to implement your solution, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Rounding Up Native Solutions

As you design a logical hierarchy, you can see which controls are provided by the UI framework and which are not. For controls that are not provided by the framework, you must create a native accessibility solution to implement those controls. In this chapter, we walked through the process of designing your implementation for those controls in UIA:

- For custom controls that map to a UIA Control Type, refer to the UI Automation Specifications and list all the Patterns and Properties necessary. If your control exhibits additional functionality other than those required by the UIA Specifications, then you must also include those Patterns and Properties in your table.
- For custom controls that do not map to a UIA Control Type, you must identify and map the functionality to Control Patterns or Properties that best exhibits the functionality of your custom control and list those requirements in your implementation table.

Methods and Events are required for completing your UIA implementation. Although you still need to specify how you will implement Methods and Events, the UIA Specifications detail which Methods and Events are required for the specific Control Patterns and Properties.

Implementation for each custom control varies, so after designing the native solutions for your custom controls, refer to the MSDN Web site on how to take your custom controls from the design stage to actually implementing them in your product. The next chapter provides a more in-depth discussion about testing the programmatic access and keyboard access of your implementation and delivery of your product.

Go further: For common frameworks and their accessibility guidelines, go to http://go.microsoft.com/fwlink/?LinkId=150842.