Chapter 4

Testing and Delivery

In our final chapter, we end with a discussion on testing the programmatic access of the UI and the keyboard access in your product. Testing for these two things can be done through a combination of software test tools, manual testing, and user scenario testing with assistive technology (AT) devices. In addition, we discuss documenting your implementation for delivery and summarize our recommendation for incorporating accessibility into your product in seven steps.

When it comes time to test your product, you want to focus on the most critical requirements or scenarios for your product first. For software that is complex, focus on the parts that are most critical to your scenarios or are most commonly used (for example, the Start menu in Windows). Once your core scenarios have been tested and verified, you can move onto any secondary requirements or scenarios.

Programmatic access and keyboard access are two critical requirements for accessibility. Without them, many different users of AT (such as screen reader and on-screen keyboard users) would be affected and would not be able to use your product at all.

To test programmatic access that is designed using UI Automation (UIA) on a Windows platform, Microsoft offers two types of test tools: (1) investigation tools and (2) a UIA testing framework called UIA Verify. Investigation tools are manual, ad-hoc test tools that allow you to quickly check the UI's underlying structure and properties. Investigation tools can also help you check the implementation of your logical hierarchy as well. UIA Verify, on the other hand, provides automated testing, where the framework has the ability to integrate into the test code and conduct regular, automated testing or spot checks of UIA test scenarios. The goal of the test framework is to promote consistent implementation across products and platforms (even those other than the Windows operating system). Because the source code is available for the framework, the code can be ported or enhanced for more advanced testing scenarios.

In addition to verifying the programmatic access, some of these tools can help you assess the implementation of your keyboard access, but, as you will learn, the tools can only go so far. So, it is important to manually verify that all of your scenarios can be accomplished with only the keyboard.

Although test tools can aid in confirming that your implementation meets the UIA Specification, ultimately, your end user's experience is what's vital to your product's success. Not only should the "nuts and bolts" of your application work and meet your users' needs as expected, but it should also be easy and intuitive for them to use, as well. In addition to obtaining feedback from a public beta release, observe users' overall experiences with your product through usability testing. You can also do heuristic evaluations internally by having employees within your company try your product and give you feedback. Because accessibility shares many requirements and best practices with many usability and UI design guidelines, you can focus on important user scenarios that impact many more users than you might have thought.

Accessibility Testing and Test Automation

While programmatic access to the UI is crucial for making software accessible today, the implementation for it is often reused by automated test tools and ATs in many different ways. Screen readers, for instance, announce desktop actions and keyboard input in speech recognition programs. On the other hand, automated test tools would use the accessibility API support for hit testing. Because of the diverse use of the accessibility API support, conflicts of interest can occur.

Before UIA, test automation used Microsoft Active Accessibility (MSAA), properties, such as `accName`, as unique and persistent identifiers to keep track of UI elements on-screen. The `Name` property was never intended to be used as a unique identifier among siblings, and using it as such can lead to unwanted results, polluting the accessibility object model by rendering a non-"human readable" string. The same rule applies to invisible or layout elements in the accessibility objects. The `Name` property should never be given a value of "MyAppHost," for instance, even if it is a layout object that is invisible to the users, or screen reader users may hear "MyAppHost" somewhere in your application. With UIA, a few new properties such as `AutomationId`, `RuntimeId`, and `ClassName` are introduced to help identify objects among siblings.

Go further: For UIA Properties and definitions go to http://go.microsoft.com/fwlink/?LinkId=150842.

Tools

For programmatic and keyboard requirements, there is no one tool that can verify your full implementation. Investigation tools and the UIA Verify framework are complementary. Investigation tools will allow you to manually check your implementation, while UIA Verify is automated and apply heuristics to help you verify that your implementation meets UIA Specification requirements. For keyboard access, manual testing should also be used to ensure access works for all navigation and user scenarios.

Depending on your control framework, there may be a variety of tools you may need to use for testing. The tools that we introduce are available on the Windows platform and can test UIA implementations. Regardless of the tools you use, remember that tools are only indicators your implementation may be wrong (or right). Try to use a variety of tools to verify your implementation and, when possible, find users of ATs, such as screen readers, to use your UI.

Investigation Tools

Investigation tools are manual test tools that allow you to quickly assess the UI for incorrect programmatic access implementations.

Inspect Objects (Inspect) and UI Spy are two investigation tools in the Microsoft Windows Software Development Kit (SDK) that provide a view of the programmatic implementation for the UI that uses a Windows Automation API, such as MSAA or UIA. They allow you to view the UI's underlying structure and properties, as well as interact with the elements, but they will only show you what was implemented and not indicate where your implementation is incorrect. As a result, you must understand the UI and all aspects of the accessibility framework that your product is built on, as well the output of results coming from the tool. Table 4-1 lists the pros and cons of these tools.

TABLE 4-1 Pros and cons of investigation tools

Pros	Cons
• Allows you to quickly investigate a UI. • Provides a raw view of the programmatic access in your product.	• You must understand the UI as well as output results from the tool. • Does not point out if there are problems with your implementation; you must rely on your knowledge to resolve any issues.

UI Spy also offers logging for UIA Events by types, as well as by scope. For instance, UI Spy can listen for `StructureChanged` UIA Events coming from a specific dialog box.

Accessible Event Watcher (AccEvent) is another investigation tool that will help you to assess your programmatic access. AccEvent is included in the MSAA SDK and allows you to review the WinEvents raised by the Windows Automation API.

Go further: For the Microsoft Windows SDK, go to http://go.microsoft.com/fwlink/?LinkId=150842.

UIA Verify Test Automation Framework

Intended to verify the implementation of the Windows Automation API, UIA Verify is a suite of test libraries that will help you test your UIA Provider implementations.

Using UIA Verify, you can write an automated test driver that runs a set of UIA test scenarios per the UIA Specification. You can also use the visual, front-end UI to run spot tests on built-in test scenarios. The tool will report the test results in XML or HTML format, and you can use that as a source for investigation requirements. Not all errors are obvious, and some errors suggest checking the validity of the problem. For example, because we rarely see a button control that can be accessible without a name, the test will report an error if your button control is left without a `Name` Property.

When UIA Verify alerts you to an error, use an investigation tool to look at the issue. Does the error seem reproducible? Visual UIA Verify, the front-end GUI of UIA Verify, can be handy to re-run the test with specific UI elements on screen. For some types of issues, you may need to use other investigation tools, such as Inspect or AccEvent, to keep track of object information at run time in greater details.

UIA Verify provides bugs about your accessible implementation, but their results are not conclusive. For instance, suppose you had a button visually labeled "OK." If you set its `Name` property to "Cancel," UIA Verify would only recognize that a button should have a programmatic name, but it would not be able to verify that the name is correct. In this case, UIA Verify would not raise an error. Final confirmation that the accessibility name matched the exact UI text on-screen would have to be done visually (optical character recognition technology may help to resolve such issues in the future, but it is still difficult to get to 100 percent accuracy as of today). Table 4-2 lists the pros and cons of UIA Verify.

TABLE 4-2 Pros and cons of the UIA Verify Test Automation Framework

Pros	Cons
• Can identify problems in your implementation. • Provides recommendations on how to them. • Can quickly give a rough idea on how well your implementation is working.	• Cannot fully review your implementation. • Errors indicated by UIA Verify are not conclusive.

Table 4-3 provides a summary and resource links for the investigation and verification test tools mentioned for testing the programmatic access and keyboard access of Win32 applications.

TABLE 4-3 Tools for Testing Programmatic and Keyboard Access

Tool	Description
Inspect Objects (Inspect)	Investigation tool that allows you to examine the element's patterns and properties as well as navigate the tree. Inspect allows you to interact with the elements through the accessibility APIs and navigate the elements by keyboard, mouse, or navigation methods provided by the framework.
Accessible Event Watcher (AccEvent)	Investigation tool that allows you to review events raised by the Windows Automation API. You can scope the events you want to listen to, the properties that should be included with those events, and which window to listen to for the events.
UI Spy	Investigation tool that allows you to examine the UIA Tree, Elements, and Events. UI Spy enables developers and testers to view and interact with the user interface (UI) elements of an application. By viewing the application's UI hierarchical structure, Property values, and raised Events, developers and testers can verify that the UI they are creating is programmatically accessible to assistive technology devices such as screen readers.
UI Automation Verify (UIA Verify) Test Automation Framework	Verification tool that checks your implementation at run time to confirm whether the UIA Provider is implementing correct tree, Patterns, and Properties. The UIA Verify facilitates manual and automated testing of the UIA Providers.

Go further: For more information and to download test tools, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Keyboard

Because all applications must be navigable using only a keyboard, be sure to test your keyboard access. Try unplugging your mouse, and use only a keyboard to access all the functionality of your software. Ensure that the navigation via keyboard follows the order of controls that need keyboard focus.

Go further: For more information on testing keyboard accessibility and guidelines on designing keyboard access, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Users and AT Devices

Throughout the development cycle, it is important to keep your users in mind. The earlier you can get feedback from actual users on your product, the less costly it is to incorporate their changes into your product. Although you may supplement your testing with third-party AT programs to test your work, beware that ATs can be complex, and you can very easily mis-interpret the information you receive from them. So, it's a good idea to get users of AT to interact with your application by using the AT devices to (1) alert you to problems that your test tools might have missed and (2) to assess your users' experience with your product. If an issue does arise when using AT programs, try to isolate the scenario, and analyze the cause using the test tools mentioned in this chapter.

Delivery

Once your product has gone through testing, and necessary corrections have been made, it's time to deliver your product. Make sure that your implementation is properly documented and that the documentation is available in accessible formats. In your documentation, be sure to address the following questions:

- How did you address your users' needs? What did your programmatic access provide?
- How do you use your software with a keyboard? Do you expose a new UI that may be difficult to learn without the ability to see the screen? Your users may not use a mouse, so describing how to navigate a new UI by keyboard is very valuable information.
- What is the structure and implementation of your design? While end-users may not necessarily be interested in the technical details, AT vendors would find your specification very useful for optimizing the user experience.
- What did you not implement? Explain what was not implemented and what is not supported in your accessibility documentation. Document any workarounds if available.

Go further: For examples on declarations of conformance, go to http://go.microsoft.com/fwlink/?LinkId=150842.

Conclusion: 7 Steps to a Better Computing World

We now leave you with seven steps that we recommend for incorporating accessibility into your software development lifecycle:

1. Decide if accessibility is an important aspect to your software. If it is, learn and appreciate how it enables real users to live, work, and play, to help guide your design.
2. As you design solutions for your requirements, use controls provided by your framework (standard controls) as much as possible, and avoid any unnecessary effort and costs of custom controls.
3. Design a logical hierarchy for your product, noting where the standard controls, any custom controls, and keyboard focus are in the UI.
4. Design basic accessibility system settings (such as keyboard navigation, high contrast, and high dpi) into your product, according to your framework's accessibility requirements.
5. Implement your design, using the Microsoft Accessibility Developer Center and your framework's accessibility specification as a reference point.
6. Test your product to ensure that end users will be able to take advantage of the accessibility techniques implemented in it.
7. Deliver your finished product and document your accessible implementation.

It's very easy to get lost in the details of providing accessibility in your software, but with UIA, we believe that you can create flexible and intuitive products that support accessibility. With the number of accessible technology users expected to rise to 70 million by 2010, up from 57 million in 2003 (Forrester 2004), and with more than half of computer users today that could benefit from accessible technology (Forrester 2003), creating accessible products makes good business sense and is the right thing to do. Not only are you addressing the needs of those who need it, you are working to make the experience for all of your users better.

Go further: For more information on developing accessible products and to share ideas with other accessibility developers, go to http://go.microsoft.com/fwlink/?LinkId=150842.

References

Forrester Research, Inc. 2004. "Accessible Technology in Computing: Examining Awareness, Use, and Future Potential." Cambridge, MA. 41.

——. 2003. "The Wide Range of Abilities and Its Impact on Technology." Cambridge, MA. 10.