This chapter continues the discussion of workflow persistence that began in Chapter 11. In that chapter, you learned the basics of workflow persistence using the SqlWorkflowInstanceStore. By following the examples presented in that chapter, you should now know how to enable persistence for applications that are hosted by the WorkflowApplication class as well as workflow services hosted by IIS or self-hosted by the WorkflowServiceHost class.

This chapter focuses on ways to extend or customize workflow persistence. It builds upon the examples that were presented in Chapter 11. Additional examples in this chapter extend persistence using the PersistenceParticipant class and demonstrate the promotion of properties to make them externally queryable. Another example demonstrates how to use the WorkflowControlEndpoint to manage active workflow instances.

The chapter concludes with an example that implements a custom instance store. The instance store persists workflow instances to the file system rather than to a database.

---

**Note** This chapter assumes that you are using the examples presented in Chapter 11 as the starting point for this chapter. In particular, the ActivityLibrary, ServiceLibrary, ServiceHost, and OrderEntryConsoleClient projects that were first developed in Chapter 11 will be used in this chapter.

---

**Understanding the PersistenceParticipant Classes**

You can customize workflow persistence in two primary ways. First, you can implement your own instance store. This option provides you with complete flexibility as to how persistence is implemented. However, this is also the most labor-intensive option. Second, WF provides the PersistenceParticipant class that enables you to participate in workflow persistence without the need to implement your own instance store. By deriving a custom workflow extension from the abstract PersistenceParticipant class, you can inject additional data that is persisted along with the workflow instance.
Note Developing your own instance store is demonstrated later in this chapter.

The PersistenceParticipant Class

Here are the most important members of the PersistenceParticipant class:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CollectValues</td>
<td>A virtual method that is invoked to collect additional data to be persisted</td>
</tr>
<tr>
<td>PublishValues</td>
<td>A virtual method that is invoked to load additional data that was previously persisted</td>
</tr>
<tr>
<td>MapValues</td>
<td>A virtual method that is invoked to provide visibility into the data that was collected by all persistence participants</td>
</tr>
</tbody>
</table>

You can follow these steps to use this extension mechanism:

1. Develop a custom workflow extension that derives from the PersistenceParticipant class (found in the System.Activities.Persistence namespace).
2. Override the virtual CollectValues method to inject additional data to be persisted.
3. Optionally, override the virtual PublishValues method to load additional data that was previously persisted.
4. Optionally, override the virtual MapValues method to review data to be persisted that was collected from all persistence participants.

The CollectValues and PublishValues methods complement each other. The CollectValues method is invoked when a workflow instance is persisted and is your opportunity to add data to be persisted. The method signature defines two out arguments of type IDictionary that must be populated by your code. One dictionary is used for read-write values and the other for write-only values. The difference between the two collections is that read-write values are expected to make a round-trip. They are persisted and then loaded when the workflow instance is loaded. The write-only values are persisted but not loaded. By specifying them as write-only values, you are indicating that they are not vital to the successful execution of the workflow. They might be queried and used by other nonworkflow portions of the application.

The PublishValues method is invoked during the process of loading a workflow instance that was previously persisted. The method is passed an IDictionary of read-write values that were previously persisted. This is your opportunity to retrieve each named value from the collection and load it back into memory.

Each value that you persist or load is uniquely identified with a string name that you must provide. The name is defined as an XName so it includes a full namespace.

The other virtual method that you can choose to override is MapValues. The purpose of this method is not as straightforward as the other methods. To better understand its purpose, you need to understand that persistence is accomplished in stages. In the first stage, the CollectValues method is