WebSphere® Application Server

Using the WorkArea Facility

Version 4.0

GC09-4550-00

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 23.

First Edition (March 2001)

Order publications through your IBM representative or through the IBM branch office serving your locality.

Contents

Figures

Using the WorkArea Facility

Introduction

One of the foundations of distributed computing is the ability to pass information, typically in the form of arguments to remote methods, from one process to another. When application-level software is written over middleware services, many of the services rely on information beyond that passed in the application's remote calls. Such services often make use of the implicit propagation of private information in addition to the arguments passed in remote requests; the two most typical users of such a feature are security and transaction services. Security certificates or transaction contexts are passed without the knowledge or intervention of the user or application developer. The implicit propagation of such information means that application developers do not have to manually pass the information in method invocations, which makes programming less error-prone, and the services requiring the information do not have to expose it to application programmers. Information like security credentials can remain secret.

The WebSphere work-area facility gives application programmers a similar facility. Applications can create a work area, insert information into it, and make remote invocations. The work area is propagated with each remote method invocation, eliminating the need to explicitly include an appropriate argument in the definition of every method. The methods on the server side can use or ignore the information in the work area as appropriate. If methods in a server receive a work area from a client and subsequently invoke other remote methods, the work area is transparently propagated with the remote requests. When the creating application is done with the work area, it terminates it.

There are two prime considerations in deciding whether to pass information explicitly as an argument or implicitly by using a work area. These considerations are:

- Pervasiveness: Is the information used in a majority of the methods in an application?
- Size: Is it reasonable to send the information even when it will not be used?

When information is sufficiently pervasive that it is easiest and most efficient to make it available everywhere, application programmers can use the work-area facility to simplify programming and maintenance of code. The argument does not need to go onto every argument list. It is much easier to put the value into a work area and propagate it automatically. This is

especially true for methods that simply pass the value on but do nothing with it. Methods that make no use of the propagated information simply ignore it.

Work areas can hold any kind of information, and they can hold an arbitrary number of individual pieces of data, each stored as a property.

Structure of work areas

The information in a work area consists of a set of properties; a *property* consists of a key-value-mode triple. The key-value pair represents the information contained in the property; the key is a name by which the associated value is retrieved. The mode determines whether the property can be removed or modified.

Property modes

There are four possible mode values for properties, as shown in Figure 1:

```
public final class PropertyModeType {
   public static final PropertyModeType normal;
   public static final PropertyModeType read_only;
   public static final PropertyModeType fixed_normal;
   public static final PropertyModeType fixed_readonly;
};
```

Figure 1. Code example: The PropertyModeType definition

A property's mode determines three things:

- Whether the value associated with the key can be modified
- Whether the property can be deleted
- Whether the mode associated with the key-value pair can be modified

The two read-only modes forbid changes to the information in the property; the two fixed modes forbid deletion of the property.

The work-area facility does not provide methods specifically for the purpose of modifying the value of a key or the mode associated with a property. To change information in a property, applications simply rewrite the information in the property; this has the same effect as updating the information in the property. The mode of a property governs the changes that can be made. "Modifying key-value pairs" describes the restrictions each mode places on modifying the value and deleting the property. "Changing modes" on page 3 describes the restrictions on changing the mode.

Modifying key-value pairs

Each property can have one of four modes, which determine how the property can be manipulated. There are two kinds of changes governed by the modes:

- Modifications to the property, either by changing property in the originating work area or masking it in a nested work area (see "Nested work areas" on page 4 for more information on nesting work areas)
- Deletion of the property

The four modes and their characteristics follow:

- Normal: The value of the key can be modified; if the key is present, it can take any value. The property can be deleted. This is the default mode.
- Read-only: The value of the key cannot be modified; if the key is present, the associated value must be the originally set value. The property can be deleted.
- Fixed normal: The key must be present, but the value of the key can be modified. The property cannot be deleted.
- Fixed read-only: The key must be present, and the value of the key cannot be modified. The property cannot be deleted.

The two read-only modes forbid changes to the information in the property; the two fixed modes forbid deletion of the property.

The mode is determined when the property is inserted into a work area. The default mode is normal, allowing modification and deletion of the property.

Changing modes

The mode associated with a property can be changed only according to the restrictions of the original mode. The read-only and fixed read-only properties do not permit modification of the value or the mode. The fixed normal and fixed read-only modes do not allow the property to be deleted. This set of restrictions leads to the following permissible ways to change the mode of a property within the lifetime of a work area:

- If the current mode is normal, it can be changed to any of the other three modes: fixed normal, read-only, fixed read-only.
- If the current mode is fixed normal, it can be changed only to fixed read-only.
- If the current mode is read-only, it can be changed only by deleting the property and re-creating it with the desired mode.
- If the current mode is fixed read-only, it cannot be changed.
- If the current mode is not normal, it cannot be changed to normal. If a property is set as fixed normal and then reset as normal, the value is updated but the mode remains fixed normal. If a property is set as fixed normal and then reset as either read-only or fixed read-only, the value is updated and the mode is changed to fixed read-only.

Note: The key, value, and mode of *any* property can be effectively changed by terminating the work area in which the property was created and

creating a new work area. Applications can then insert new properties into the work area. This is not precisely the same as changing the value in the original work area, but some applications can use it as an equivalent mechanism.

Nested work areas

Applications can nest work areas. When an application creates a work area, a work-area context is associated with the creating thread. If the application thread creates another work area, the new work area is nested within the existing work area and becomes the current work area. Nested work areas allow applications to define and scope properties for specific tasks without having to make them available to all parts of the application. All properties defined in the original, enclosing work area are visible to the nested work area. The application can set additional properties within the nested work area that are not part of the enclosing work area.

An application working with a nested work area does not actually see the nesting of enclosing work areas. The current work area appears as a flat set of properties that includes those from enclosing work areas. In Figure 2 on page 5, the enclosing work area holds several properties and the nested work area holds additional properties. From the outermost work area, the properties set in the nested work are not visible. From the nested work area, the properties in both work areas are visible.

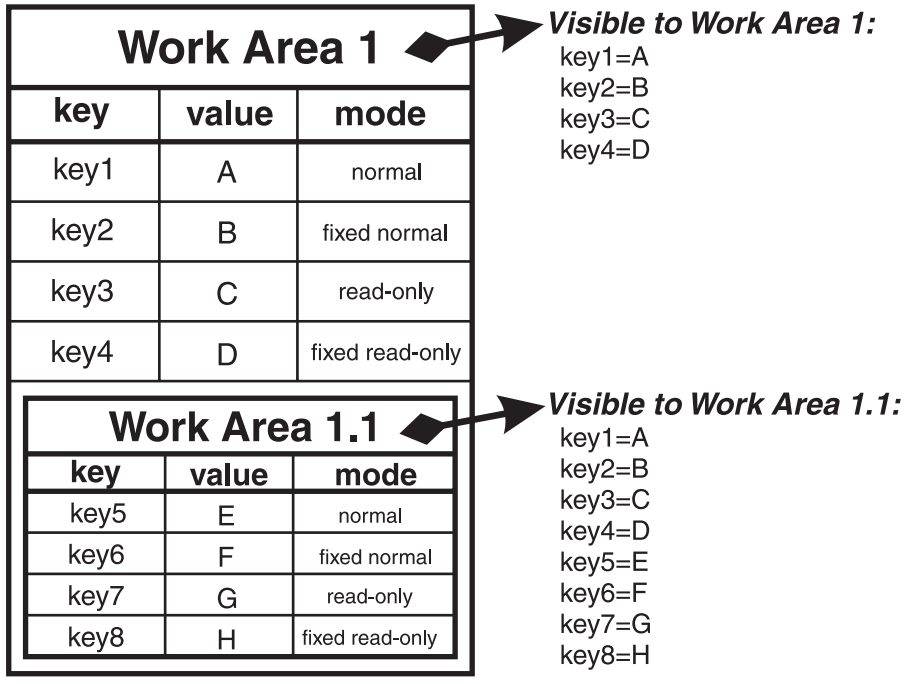

Figure 2. Defining new properties in nested work areas

Nesting can also affect the apparent settings of the properties. Properties can be deleted from or directly modified only within the work areas in which they were set, but nested work areas can also be used to temporarily override information in the property without having to modify the property. Depending on the modes associated with the properties in the enclosing work area, the modes and the values of keys in the enclosing work area can be overridden within the nested work area.

The mode associated with a property when it is created determines whether nested work areas can override the property. From the perspective of a nested work area, the property modes used in enclosing work areas can be grouped as follows:

- Modes that permit a nested work area to override the mode or the value of a key locally. The modes that permit overriding are:
  - Normal
  - Fixed normal
- Modes that do not permit a nested work area to override the mode or the value of a key locally. The modes that do not permit overriding are:
  - Read-only
  - Fixed read-only

If an enclosing work area defines a property with one of the overridable modes, a nested work area can specify a new value for the key or a new mode for the property. The new value or mode becomes the value or mode seen by subsequently nested work areas. Changes to the mode are governed by the restrictions described in "Changing modes" on page 3. If an enclosing work area defines a property with one of the modes that cannot be overridden, no nested work area can specify a new value for the key.

A nested work area can delete properties from enclosing work areas, but the changes persist only for the duration of the nested work area. When the nested work area is completed, any properties that were added in the nested area vanish and any properties that were deleted from the nested area are restored.

Figure 3 illustrates the overriding of properties from an enclosing work area. The nested work area redefines two of the properties set in the enclosing work area. The other two cannot be overridden. The nested work area also defines two new properties. From the outermost work area, the properties set or redefined in the nested work are not visible. From the nested work area, the properties in both work areas are visible, but the values seen for the redefined properties are those set in the nested work area.

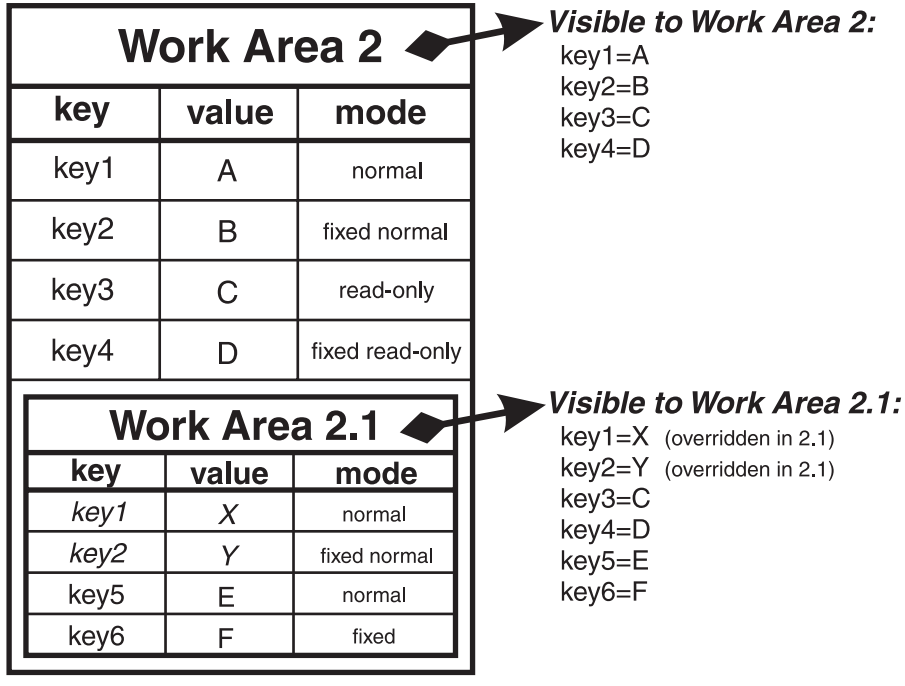

Figure 3. Redefining existing properties in nested work areas

Distributed work areas

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, by creating additional nested work areas; this information will be propagated to any remote objects it invokes. However, no changes made to a nested work area on a target object are propagated back to the calling object. The caller's work area is unaffected by changes made in the remote method.

Administration

The work-area facility is administered by using the WebSphere Application Server administrative console and the thin client administrative tool. There are two administrative tasks associated with work areas:

- Enabling or disabling use of the work-area facility
- Managing the size of the work-area information

The work-area facility can be used by clients and servers, and it must be enabled separately for each. On the server side, the use of work areas is enabled by default. To enable or disable the use of work areas for a server by using the administrative console, locate the server's configuration information and perform these steps:

1. Select the **Custom** tab on the server configuration window.
2. Select the **WorkArea** service and click the **Edit** button.
3. Click the **Enabled** box to either enable or disable the work-area service. The service is enabled if the box is checked and disabled if the box is not checked.

On the client side, the use of work areas is also enabled by default. To enable or disable the use of work areas for a client, set the com.ibm.websphere.workarea.enabled property to TRUE or FALSE before starting the client. This can be done in several ways. For example, edit the launchClient script in the ${WAS_HOME}/bin directory and add the following to the Java invocation: `-Dcom.ibm.websphere.workarea.enabled=false`.

Applications can set maximum sizes on each work area that can be sent and that can be accepted. By default, the maximum that is sent by a client and accepted, and possibly re-sent, by a server is 32,768 bytes.

To change the size of a work area that can be handled by a server, or accepted to or in a server, locate the server's configuration information by using the administrative console and perform these steps:

1. Select the **Custom** tab on the server configuration window.
2. Select the **WorkArea** service and click the **Edit** button.
3. Enter a new value for **maxSendSize** field to modify the size of the work area the server can send, or enter a new value for **maxReceiveSize** field to modify the size of the work area the server can accept.

To change the size of a work area than can be sent from a client, set the com.ibm.websphere.workarea.maxSendSize property to the desired number of bytes before starting the client. This can be done in several ways. For example, to set the maximum size to 10,000 bytes, edit the launchClient script in the ${WAS_HOME}/bin directory and add the following to the Java invocation: `-Dcom.ibm.websphere.workarea.maxSendSize=10000`.

The maximum size that can be specified is determined by the maximum value expressible in the Java Integer data type, 2,147,483,647. The smallest maximum size that can be specified is 1. Using a maximum size of 1 byte effectively means that no requests associated with the work area can leave the system or enter another system. A value of 0 means that no limit is imposed. A value of -1 means that the default value is to be honored. The default value is also used if a invalid value or a malformed property is specified.

Running work-area applications

The work-area service is available to any Java™ 2 Platform Enterprise Edition (J2EE) web module, enterprise-bean module, or client. To compile applications that use the work-area facility, ensure that the acwa.jar file is on the classpath.

The example application

This document uses a simple application to illustrate the use of the work-area facility. In this example, the client creates a work area and inserts two properties into the work area: a site identifier and a priority. The site-identifier is set as a read-only property; the client does not allow recipients of the work area to override the site identifier. This property consists of the key `company` and a static instance of a SimpleSampleCompany object.The priority property consists of the key `priority` and a static instance of a SimpleSamplePriority object. The object types are defined as shown in Figure 4 on page 9.

```
public static final class SimpleSampleCompany {
   public static final SimpleSampleCompany Main;
   public static final SimpleSampleCompany NewYork_Sales;
   public static final SimpleSampleCompany NewYork_Development;
   public static final SimpleSampleCompany London_Sales;
   public static final SimpleSampleCompany London_Development;
}

public static final class SimpleSamplePriority {
   public static final SimpleSamplePriority Platinum;
   public static final SimpleSamplePriority Gold;
   public static final SimpleSamplePriority Silver;
   public static final SimpleSamplePriority Bronze;
   public static final SimpleSamplePriority Tin;
}
```

Figure 4. Code example: The SimpleSampleCompany and SimpleSamplePriority classes

The client then makes an invocation on a remote object. The work area is automatically propagated; none of the methods on the remote object take a work-area argument. On the remote side, the request is first handled by the SimpleSampleBean; the bean first reads the site-identifier and priority properties from the work area. The bean then intentionally attempts, and fails, both to write directly into the imported work area and to override the read-only site-identifier property.

The SimpleSampleBean successfully begins a nested work area, in which it overrides the client's priority, then calls another bean, the SimpleSampleBackendBean. The SimpleSampleBackendBean reads the properties from the work area, which contains the site identifier set in the client and priority set in the SimpleSampleBean. Finally, the SimpleSampleBean completes its nested work area, writes out a message based on the site-identifier property, and returns.

The implementation of this application is discussed in "Writing the example application" on page 10.

Special considerations

Programmers who use work areas must take into consideration two concerns that can arise. The first is related to interoperability between the Enterprise JavaBeans and CORBA programming models, and the second is related to threading.

Although the work-area facility can be used across the Enterprise JavaBeans and the CORBA programming models, many composed data types cannot be successfully utilitized across those boundaries. For example, if a SimpleSampleCompany instance is passed from the WebSphere environment into a CORBA environment, the CORBA application can retrieve the

SimpleSampleCompany object encapsulated within a CORBA Any object from the work area, but it cannot extract the value from it. Likewise, an IDL-defined struct defined within a CORBA application and set into a work area will not be readable by an application using the UserWorkArea class. Applications can avoid this incompatibility by directly setting only primitive types, like integers and strings, as values in work areas, or by implementing complex values with structures designed to be compatible, like CORBA valuetypes. Also, CORBA Anys that contains either the tk_null or tk_void typecode can be set into the work area by using the CORBA interface, but the work-area specification cannot allow the J2EE implementation to return null on a lookup that retrieves these CORBA-set properties without incorrectly implying that there is no value set for the corresponding key. If a J2EE application tries to retrieve CORBA-set properties that are non-serializable, or contain CORBA nulls or void references, the com.ibm.websphere.workarea.IncompatibleValue exception is raised.

Work areas must be used cautiously in applications that use the Java's Abstract Windowing Toolkit. The ATW implementation is multithreaded, and work areas begun on one thread are not available on another. For example, if a program begins a work area in response to an AWT event, like pressing a button, the work area may not be available to any other part of the application after the execution of the event completes.

Writing the example application

Applications interact with the work-area facility by using the UserWorkArea interface. This interface, shown in Figure 5, defines all the methods used to create, manipulate, and terminate work areas.

```
package com.ibm.websphere.workarea;

public interface UserWorkArea {
   void begin(String name);
   void complete() throws NoWorkArea, NotOriginator;

   String getName();
   String[] retrieveAllKeys();
   void set(String key, java.io.Serializable value)
      throws NoWorkArea, NotOriginator, PropertyReadOnly;
   void set(String key, java.io.Serializable value, PropertyModeType mode)
      throws NoWorkArea, NotOriginator, PropertyReadOnly;
   java.io.Serializable get(String key);
   PropertyModeType getMode(String key);
   void remove(String key)
      throws NoWorkArea, NotOriginator, PropertyFixed;
}
```

Figure 5. Code example: The UserWorkArea interface

Note: Enterprise JavaBeans applications can use the UserWorkArea interface only within the implementation of methods in the remote interface; likewise, servlets can use the interface only within the service method of the HTTPServlet class. Use of work areas within any lifecycle method of a servlet or enterprise bean is not supported by the work-area specification and is considered a deviation from the work-area programming model.

The work-area facility defines the following exceptions for use with the UserWorkArea interface:

- NoWorkArea: thrown when a request requires an associated work area but none is present.
- NotOriginator: raised when a request attempts to manipulate the contents of an imported work area.
- PropertyReadOnly: raised when a request attempts to modify a read-only or fixed read-only property.
- PropertyFixed: raised by the remove method when the designated property has one of the fixed modes.

Creating a work area

The client side of the application described in "The example application" on page 8 creates a work area and inserts the site-identifier and priority properties into the work area. This requires four steps on the part of the client:

1. Binding to the work-area facility
2. Creating a new work area
3. Inserting information into the work area
4. Terminating the work area when it is no longer needed

Binding to the work-area facility

The work-area facility provides a JNDI binding to an implementation of the UserWorkArea interface under the name java:comp/websphere/UserWorkArea. Applications that need to access the service can perform a lookup on that JNDI name, as shown in Figure 6 on page 12.

```
import com.ibm.websphere.workarea.*;
import javax.naming.*;

public class SimpleSampleServlet {
  ...

  InitialContext jndi = null;
  UserWorkArea userWorkArea = null;
  try {
     jndi = new InitialContext();
     userWorkArea = (UserWorkArea)jndi.lookup(
        "java:comp/websphere/UserWorkArea");
  }
  catch (NamingException e) { ... }
}
```

Figure 6. Code example: Binding to the work-area facility

Beginning a work area

After a client has a reference to the UserWorkArea interface, it can use the begin method to create a new work area and associate it with the calling thread. The begin method takes a string as an argument; the string is used to name the work area. The argument must not be null, which causes the java.lang.NullPointer exception to be raised. In Figure 7, the application begins a new work area with the name `SimpleSampleServlet`.

```
public class SimpleSampleServlet {
...
   try {
      ...
      userWorkArea = (UserWorkArea)jndi.lookup(
         "java:comp/websphere/UserWorkArea");
   }
   ...

   userWorkArea.begin("SimpleSampleServlet");
   ...
}
```

Figure 7. Code example: Creating a new work area

Each work area must also be terminated within the process that created it; each call to the begin method must have a corresponding call to the complete method. See “Completing a work area” on page 15 for more information.

The begin method is also used to create nested work areas; if a work area is associated with a thread when the begin method is called, the method creates a new work area nested within the existing work area.

The work-area facility makes no use of the names associated with work areas; programmers can name work areas in any way they choose. Names are not required to be unique, but the usefulness of the names for debugging is enhanced if the names are distinct and meaningful within the application.

Applications can use the getName method to return the name associated with a work area by the begin method.

Setting properties in a work area

An application with a current work area can insert properties into the work area and retrieve the properties from the work area. The UserWorkArea interface provides two set methods for setting properties and a get method for retrieving properties. The two-argument set method inserts the property with the property mode of normal. The three-argument set method takes a property mode as the third argument. See "Setting property modes" on page 14 for more information on specifying property modes.

Both set methods take the key and the value as arguments. The key is a String; the value is an object of the type java.io.Serializable. None of the arguments can be null, which causes the java.lang.NullPointer exception to be raised.

The property classes used in the example application: The example application uses objects of two classes, the SimpleSampleCompany class and the SimpleSampleProperty class, as values for properties. The SimpleSampleCompany class is used for the site identifier, and the SimpleSamplePriority class is used for the priority. These classes are shown in Figure 4 on page 9.

```
public class SimpleSampleServlet {
   ...
   userWorkArea.begin("SimpleSampleServlet");

   try {
      // Set the site-identifier (default is Main).
      userWorkArea.set("company",
         SimpleSampleCompany.Main, PropertyModeType.read_only);

      // Set the priority.
      userWorkArea.set("priority", SimpleSamplePriority.Silver);
   }

   catch (PropertyReadOnly e) {
     // The company was previously set with the read-only or
     // fixed read-only mode.
     ...
   }

    catch (NotOriginator e) {
     // The work area originated in another process,
     // so it can't be modified here.
     ...
   }

   catch (NoWorkArea e) {
      // There is no work area begun on this thread.
      ...
   }

   // Do application work.
   ...
}
```

Figure 8. Code example: Setting properties in a work area

The get method takes the key as an argument and returns a Java Serializable object as the value associated with the key. For example, to retrieve the value of the `company` key from the work area, Figure 8 uses the get method on the work area to retrieve the value.

Setting property modes: The two-argument set method on the UserWorkArea interface takes a key and a value as arguments and inserts the property with the default property mode of normal. To set a property with a different mode, applications must use the three-argument set method, which takes a property mode as the third argument. The values used to request the property modes follow:

- Normal: PropertyModeType.normal
- Fixed normal: PropertyModeType.fixed_normal
- Read-only: PropertyModeType.read_only

- Fixed read-only: PropertyModeType.fixed_readonly

(See Figure 1 on page 2 for more information.)

Completing a work area

After an application has finished using the work area, it can terminate the work area by calling the complete method on the UserWorkArea interface. This terminates the association with the calling thread and destroys the work area. If the complete method is called on a nested work area, the nested work area is terminated and the parent work area becomes the current work area. If there is no work area associated with the calling thread, the NoWorkArea exception is thrown.

Every work area must be terminated, and work areas can be terminated only by the originating process. For example, if a server attempts to call the complete method on a work area that originated in a client, the work-area NotOriginator exception is thrown. Figure 9 shows the termination of the work area created in the client application.

```
public class SimpleSampleServlet {
   ...
   userWorkArea.begin("SimpleSampleServlet");
   userWorkArea.set("company",
       SimpleSampleCompany.Main, PropertyModeType.read_only);
   userWorkArea.set("priority", SimpleSamplePriority.Silver);
   ...

   // Do application work.
   ...

   // Terminate the work area.
   try {
      userWorkArea.complete();
   }

   catch (NoWorkArea e) {
      // There is no work area associated with this thread.
      ...
   }

   catch (NotOriginator e) {
      // The work area was imported into this process.
      ...
   }
  ...
}
```

Figure 9. Code example: Terminating the work area

Using a work area

The server side of the application described in "The example application" on page 8 accepts remote invocations from clients. With each remote call, the server also gets a work area from client if the client has created one. The work area is propagated transparently. None of the remote methods includes the work area on its argument list.

In the example application, the server objects utilize the work-area interface for demonstration purposes only. For example, the SimpleSampleBean intentionally attempts to write directly to an imported work area, which triggers the NotOriginator exception. Likewise, the bean intentionally attempts to mask the read-only SimpleSampleCompany, which triggers the PropertyReadOnly exception. The SimpleSampleBean also nests a WorkArea and successfully overrides the priority property before invoking the SimpleSampleBackendBean. A real business application would extract the work area properties and use them to guide the local work. The SimpleSampleBean mimics this by writing a message that function is denied when a request emanates from a sales environment.

The server must bind to the work-area facility before it can manipulate information in work areas.

Binding to the work-area facility

The work-area facility provides a JNDI binding to an implementation of the UserWorkArea interface under the name java:comp/websphere/UserWorkArea. Applications that need to access the service can perform a lookup on that JNDI name, as shown in "Binding to the work-area facility" on page 11.

Extracting the name of the active work area

Applications use the getName method on the UserWorkArea interface to retrieve the name of the current work area. This is the recommended method for determining whether the thread is associated with a work area; if the thread is not associated with a work area, the getName method will return null. Figure 10 on page 17 uses the getName method on the work area to retrieve the name of the active work area; in this example, the name of the work area corresponds to the name of the class in which the work area was begun.

```
public class SimpleSampleBeanImpl implements SessionBean {

    ...

    public String [] test() {
       // Get the work-area reference from JNDI.
       ...

       // Retrieve the name of the work area. In this example,
       // the name is used to identify the class in which the
       // work area was begun.
       String invoker = userWorkArea.getName();
       ...
   }
}
```

Figure 10. Code example: Retrieving the name from a work area

Modifying information in a work area

Work areas are inherently associated with the process that creates them. In the sample application, the client begins a work area and sets into it the site-identifier and a priority properties into it. This work area is propagated to the server when the client makes a remote invocation.

In Figure 11, the server-side sample bean attempts to write directly to the imported work area; this action is not permitted, and the NotOriginator exception is thrown. The sample bean must begin its own work area in order to override any imported properties.

```
public class SimpleSampleBeanImpl implements SessionBean {

   public String [] test() {
     ...
     String invoker = userWorkArea.getName();

     try {
       userWorkArea.set("key", "value");
     }
     catch (NotOriginator e) {
     }
     ...
  }
}
```

Figure 11. Code example: Attempting to modify an imported work area

Nesting work areas: Applications nest work areas in order to temporarily override properties imported from a client process. The nesting mechanism is

automatic; invoking begin on the UserWorkArea interface from within the scope of an existing work area creates a nested work area that inherits the properties from the enclosing work area. Properties set into the nested work area are strictly associated with the process in which the work area was begun; the nested work area must be completed within the process that created them. If a work area is not completed by the creating process, the work-area facility terminates the work area when the process exits. After a nested work area is completed, the original view of the enclosing work area is restored. However, the view of the complete set of work areas associated with a thread cannot be decomposed by downstream processes. Figure 12 demonstrates beginning a nested work area, using the name of the creating class to identify the nested work area.

```
public class SimpleSampleBeanImpl implements SessionBean {

   public String [] test() {
      ...
      String invoker = userWorkArea.getName();
      try {
        userWorkArea.set("key", "value");
      }
      catch (NotOriginator e) {
      }

      // Begin a nested work area. By using the name of the creating
      // class as the name of the work area, we can avoid having
      // to explicitly set the name of the creating class in
      // the work area.
      userWorkArea.begin("SimpleSampleBean");

      ...
   }
}
```

Figure 12. Creating a nested work area

Applications set properties into a work area using property modes in ensure that a particular property is fixed (not removable) or read-only (not overrideable) within the scope of the given work area. In the sample application, the client sets the site-identifier property as read-only; that guarantees that the request will always be associated with the client's company identity. A server cannot override that value in a nested work area. In Figure 13 on page 19, the SimpleSampleBean attempts to change the value of the site-identifier property in the nested work area it created.

```
public class SimpleSampleBeanImpl implements SessionBean {

  public String [] test() {
      ...

      String invoker = userWorkArea.getName();
      try {
        userWorkArea.set("key", "value");
      }
      catch (NotOriginator e) {
      }

      // Begin a nested work area.
      userWorkArea.begin("SimpleSampleBean");

      try {
        userWorkArea.set("company",
                         SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }
      ...
  }
}
```

Figure 13. Code example: Attempting to modify a non-overridable property

Extracting properties from a work area

Properties can be retrieved from a work area by using the get method. The method is intentionally light-weight; there are no declared exceptions to handle. If there is no active work area or if there is no such property set in the current work area, the get method returns null. Figure 14 on page 20 shows the retrieval of the site-identifier and priority properties by the SimpleSampleBean. Recall that one property was set into an outer work area by the client, and the other property was set into the nested work area by the server-side bean; the nesting is transparent to the retrieval of the properties.

Note: The get method can raise a NotSerializableError in the relatively rare scenario in which CORBA clients set composed data types and invote enterprise-bean interfaces.

```
public class SimpleSampleBeanImpl implements SessionBean {

    public String [] test() {
      ...

      // Begin a nested work area.
      userWorkArea.begin("SimpleSampleBean");
      try {
        userWorkArea.set("company",
                         SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }

      SimpleSampleCompany company =
         (SimpleSampleCompany) userWorkArea.get("company");
      SimpleSamplePriority priority =
         (SimpleSamplePriority) userWorkArea.get("priority");
       ...
   }
}
```

Figure 14. Code example: Retrieving properties from a work area

Completing a work area

All work areas must be completed within the process in which they were created; every ivocation of the begin method must be matched by an invocation of the complete method. Work areas created in a server process are never propagated back to an invoking client process. Figure 15 on page 21 shows the sample application completing the nested work area it created earlier in the remote invocation. The UserWorkArea reference points to the outer work area after the complete method concludes.

Note that the work area service claims full local-remote transparency. Even if two beans happen to be deployed into the same server and therefore the same JVM and process, a work area begun on an invocation from another will be completed and the bean in which the request origininated will always be in the same state after any remote call as it was before.

```
public class SimpleSampleBeanImpl implements SessionBean {

    public String [] test() {
      ...

      // Begin a nested work area.
      userWorkArea.begin("SimpleSampleBean");
      try {
        userWorkArea.set("company",
                         SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }

      SimpleSampleCompany company =
         (SimpleSampleCompany) userWorkArea.get("company");
      SimpleSamplePriority priority =
         (SimpleSamplePriority) userWorkArea.get("priority");

      // Complete all nested work areas before returning.
      try {
        userWorkArea.complete();
      }
      catch (NoWorkArea e) {
      }
      catch (NotOriginator e) {
      }
   }
}
```

Figure 15. Code example: Retrieving properties from a work area

Other methods in the UserWorkArea interface

The simple example illustrated in "Creating a work area" on page 11 and "Using a work area" on page 16 does not make use of all the methods in the UserWorkArea interface. This section describes the additional methods:

Obtaining a list of all keys

The UserWorkArea interface provides the retrieveAllKeys method for retrieving a list of all the keys visible from a work area. This method takes no arguments and returns an array of strings. This method returns null if there is no work area associated with the thread. If there is an associated work area containing no properties, the method returns an array of size 0.

Querying the mode of a property

The UserWorkArea interface provides the getMode method for determining the mode of a specific property. This method takes the property's key as an argument and returns the mode as a PropertyModeType object. (See "Setting property modes" on page 14 for more information on names of mode types.)

If the specified key does not exist in the work area, the method returns PropertyModeType.normal, indicating that the property can be set and removed without error.

Deleting a property

The UserWorkArea interface provides the remove method for deleting a property from the current scope of a work area. If the property was initially set in the current scope, then removing it deletes the property. If the property was initially set in an enclosing work area, then removing it deletes the property until the current scope is completed. When the current work area is completed, the deleted property is restored.

The remove method takes the property's key as an argument. Only properties with the modes normal and read-only can be removed. Attempting to remove a fixed property causes the PropertyFixed exception to be thrown. Attempting to remove properties in work areas that originated in other processes causes the NotOriginator exception to be thrown.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will

be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and service marks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States, other countries, or both:

Advanced Peer-to-Peer Networking
AFS
AIX
APPN
AS/400
CICS
CICS OS/2
CICS/400
CICS/6000
CICS/ESA
CICS/MVS
CICS/VSE
CICSPlex
DB2
DCE Encina Lightweight Client
DFS
Encina
IBM
IBM System Application Architecture
IMS
IMS/ESA
Language Environment
MQSeries
MVS/ESA
NetView
Open Class
OS/2
OS/390
OS/400
Parallel Sysplex
PowerPC
RACF
RAMAO
RMF
RISC System/6000
RS/6000
S/390
SAA
SecureWay
TeamConnection
Transarc
TXSeries
VSE/ESA
VTAM
VisualAge
WebSphere

Domino, Lotus, and LotusScript are trademarks or registered trademarks of Lotus Development Corporation in the United States, other countries, or both.

Tivoli is a registered trademark of Tivoli Systems, Inc. in the United States, other countries, or both.

ActiveX, Microsoft, Visual Basic, Visual C++, Visual J++, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Some of this documentation is based on material from Object Management Group bearing the following copyright notices:

Copyright 1995, 1996 AT&T/NCR
Copyright 1995, 1996 BNR Europe Ltd.
Copyright 1991, 1992, 1995, 1996 by Digital Equipment Corporation
Copyright 1996 Gradient Technologies, Inc.
Copyright 1995, 1996 Groupe Bull
Copyright 1995, 1996 Expersoft Corporation
Copyright 1996 FUJITSU LIMITED
Copyright 1996 Genesis Development Corporation
Copyright 1989, 1990, 1991, 1992, 1995, 1996 by Hewlett-Packard Company
Copyright 1991, 1992, 1995, 1996 by HyperDesk Corporation
Copyright 1995, 1996 IBM Corporation
Copyright 1995, 1996 ICL, plc
Copyright 1995, 1996 Ing. C. Olivetti &C.Sp
Copyright 1997 International Computers Limited
Copyright 1995, 1996 IONA Technologies, Ltd.
Copyright 1995, 1996 Itasca Systems, Inc.
Copyright 1991, 1992, 1995, 1996 by NCR Corporation
Copyright 1997 Netscape Communications Corporation
Copyright 1997 Northern Telecom Limited
Copyright 1995, 1996 Novell USG
Copyright 1995, 1996 02 Technolgies
Copyright 1991, 1992, 1995, 1996 by Object Design, Inc.
Copyright 1991, 1992, 1995, 1996 Object Management Group, Inc.
Copyright 1995, 1996 Objectivity, Inc.
Copyright 1995, 1996 Oracle Corporation
Copyright 1995, 1996 Persistence Software
Copyright 1995, 1996 Servio, Corp.
Copyright 1996 Siemens Nixdorf Informationssysteme AG
Copyright 1991, 1992, 1995, 1996 by Sun Microsystems, Inc.
Copyright 1995, 1996 SunSoft, Inc.
Copyright 1996 Sybase, Inc.
Copyright 1996 Taligent, Inc.

This software contains RSA encryption code.

Java Compatible™ Enterprise Edition

Other company, product, and service names may be trademarks or service marks of others.