RFC 128 Accessing Exit Values from Applications

Final

9 Pages

Abstract

This RFC describes an approach for accessing the exit value from an application launched using an Application Descriptor in Application Admin.

0 Document Information

0.1 Table of Contents

0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 8.1.

`Source code is shown in this typeface.`

0.3 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	Oct 22 2007	Initial draft using information provided by bug 433 Thomas Watson, IBM, tjwatson@us.ibm.com
	Oct 26 2007	Added timeout parameter to getExitValue Thomas Watson
	Aug 05 2008	Final clean up, removed question about where the service property should be and the security questions
Final	2 December 2008	No changes. Final for CPEG voting.

1 Introduction

The Application Admin specification is used to manage an environment with many different types of applications that are simultaneously available. In some application environments when an application is finished the application is allowed to return an exit value. Currently the Application Admin Specification does not allow access to an exit value returned by an application when it is finished.

This RFC describes an approach for accessing an exit value when an application is finished.

2 Application Domain

2.1 Terminology

Exit Value – The result returned by an application after is has finished. This value is application and application container specific

3 Problem Description

Many application environments allow applications to return an exit value. For example, when launching a process on many operating system the process will return an integer value indicating a success or failure of the execution. This is evident in the java.lang.Process#exitValue() method.

Other application environments may allow more complicated exit values for returning results from the application. For example, in the Eclipse RCP application container applications are allowed to return arbitrary Objects.

4 Requirements

1. Clients which launch an application must be able to access the exit value of an application once it has finished.
2. An application must be able to return an arbitrary value for the exit value once it has finished execution.
3. Clients must be able to determine when an application exit value is available
4. Existing applications and application containers must continue to work without modification

4.1 Use Cases

4.1.1 Launching Native Commands

In Java, native commands can be executed by calling Runtime#exec(...). This returns a Process object which can be used to track the execution of the command and retrieve exit values. This is similar to how an ApplicationHandle can be used to track an application instance in Application Admin. Some scenarios may want to register native commands as ApplicationDescriptor services. The ApplicationHandle implementation could then use the Process#waitFor method to implement the lifecycle of the ApplicationHandle. But there is no way for the client to get access to the Process#exitValue from an ApplicationHandle.

4.1.2 Eclipse RCP Applications

In the Eclipse Rich Client Platform an application container is defined for running applications defined by bundles installed in the framework. An RCP application bundle uses the extension registry to specify an application definition declaratively with an extension (a la plugin.xml). The application definition specifies an entry point to the application which is used to run the application. The entry point is a class from the application bundle which implements an interface specific to the RCP application container. This class is used by the application container to launch the application and retrieve the exit value from the application when it is finished. Applications in the RCP application container are allowed to return any object type they like as exit values.

5 Technical Solution

5.1 Method getExitValue

A new method is added to the ApplicationHandle class to get the exit value from an application instance. This method allows a timeout value to be specified. If a timeout is specified this method will block until either the application has terminated or the expiration of the timeout. If the application has terminated then the exit value is returned otherwise an ApplicationException is thrown.

The getExitValue method will not be abstract and the default implementation will throw an UnsupportedOperationException. This is necessary to allow existing applications and application container implementations to continue to work without modification or re-compilation. A new ApplicationHandle service property is added to indicate if the application instance supports exit values (application.supports.exitvalue).

Key Name	Type	Default	Description
application.supports.exitvalue	Boolean	FALSE	Specifies whether the application instance supports an exit value.

5.1.1 ApplicationException JavaDoc

5.1.1.1 APPLICATION_EXITVALUE_NOT_AVAILABLE

The exit value is not available for an application instance because the instance has not terminated.

5.1.2 ApplicationHandle JavaDoc

5.1.2.1 APPLICATION_SUPPORTS_EXITVALUE

String org.osgi.service.application.ApplicationHandle.APPLICATION_SUPPORTS_EXITVALUE = "application.supports.exitvalue"

The property key for the supports exit value property of this application instance.

5.1.2.2 GetExitValue

Object org.osgi.service.application.ApplicationHandle.getExitValue(long timeout)

Returns the exit value for the application instance. The timeout specifies how the method behaves when the application has not terminated. A negative, zero or positive value may be used.

- negative - The method does not wait for termination. If the application has not terminated then an `ApplicationException` is thrown
- zero - The method waits until the application has terminated
- positive - The method waits until the application has terminated or the timeout has expired. If the timeout has expired and the application has not terminated then an `ApplicationException` is thrown.

The default implementation throws an `UnsupportedOperationException`. The application model should override this method if exit values are supported.

Parameters:
  timeout The maximum time in milliseconds to wait for the application to timeout.
Returns:
  the exit value for the application instance. The value is application specific.
Throws:
  UnsupportedOperationException if the application model does not support exit values.
  InterruptedException if the wait has been interrupted.
  ApplicationException

6 Considered Alternatives

6.1 No Block and timeout

The original proposal did not have the ability to block and wait for an application instance to terminate. If clients wanted to wait for an application to terminate they could use standard OSGi ServiceEvents to do so. But this left a rather difficult programming pattern for simple scenarios that simply wanted to launch an application and wait for the exit value. Something like the following (likely buggy code) would have to be used:

```
ApplicationHandle handle = app.launch(null);
ServiceReference[] handleRefs =
    context.getServiceReferences(
        ApplicationHandle.class.getName(),
        "(service.pid=" + handle.getInstanceId() +")");
if (handleRefs != null) {
  final boolean[] unregistered = new boolean[] {false};
  ServiceTrackerCustomizer handleCustomizedTracker =
      new ServiceTrackerCustomizer() {
        public Object addingService(ServiceReference reference) {
          return reference;
        }
        public void modifiedService(ServiceReference reference, Object service)
        {
        }
        public void removedService(ServiceReference reference, Object service)
{
          synchronized (unregistered) {
            unregistered[0] = true;
            unregistered.notifyAll();
          }
        }
  };
  ServiceTracker appTracker =
    new ServiceTracker(context, handleRefs[0], handleCustomizedTracker);
  appTracker.open();
  synchronized (unregistered) {
    while (!unregistered[0]) {
      unregistered.wait();
    }
  }
  appTracker.close();
}
Object exitData = handle.getExitValue();
```

It was determined that the best thing to do would be to allow for a timeout value instead which would make the above code look like this:

```
ApplicationHandle handle = app.launch(null);
```

```
Object exitData = handle.getExitValue(0);
```

7 Security Considerations

7.1 Exit Value Sensitivity

The exit value could contain sensitive data. Clients must have ServicePermission to acquire the ApplicationDescriptor to launch the application instance or to acquire the ApplicationHandle service.

7.2 Exit Value Memory Leak

The ApplicationHandle must maintain a reference to the application value for the lifetime of the ApplicationHandle object (i.e. until it is GC'ed). ApplicationHandle objects cannot throw away the result as soon as they are unregistered. If ApplicationHandles are prevented from GC'ing and the exit value consumes large amounts of memory/resources then a drastic memory leak will occur.

8 Document Support

8.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

8.2 Author's Address

Name	Thomas Watson
Company	IBM
Address	
Voice	(512) 838 4533
e-mail	tjwatson@us.ibm.com

8.3 Acronyms and Abbreviations

8.4 End of Document