IBM® Lotus® Expeditor 6.1 Client for Desktop

Embedded transaction container and JNDI

Lotus software

@business on demand software

This presentation explains the Embedded Transaction Container and JNDI services supported by IBM Lotus Expeditor 6.1 Client for Desktops.

Goals

- Understand the Embedded Transaction Container and Java™ naming and directory interface (JNDI) services provided by IBM Lotus Expeditor 6.1 Client for Desktop

Embedded transaction container and JNDI

The goal of this presentation is to understand the Embedded Transaction Container and JNDI services supported by IBM Lotus Expeditor 6.1 Client for Desktop.

IBM

Agenda

- JNDI

- Embedded Transaction Container

The agenda of this presentation is to explain the JNDI and Embedded Transaction Container services.

Section

JNDI

Let's start with an explanation of JNDI.

IBM

JNDI Provider

JNDI

Java naming and directory interface

- Supports the JNDI API (J2SE 1.4)
  - ▶ javax.naming (access naming services)
  - ▶ javax.naming.spi (plug in naming services)
- Delivers a light weight JNDI provider
  - ▶ Simple hierarchical name space for client applications
  - ▶ No federation of other name spaces
  - ▶ Objects registered by JNDI name in JNDI registry
    - for example, java:comp/env/jdbc/dsname
  - ▶ Declarative JNDI dynamically adds and removes objects from registry
    - Implemented as Eclipse extension point
    - Enables faster platform startup time by "lazy" creation of objects on demand
  - ▶ Does not persist objects or state information across client platform restarts
- Enables applications to lookup named objects (for example, EJBs and data sources)

The client platform provides a simple Java object JNDI registry that enables applications to lookup named objects (for example, EJBs and data sources).

The JNDI provider enables a local naming directory for objects running in the client platform to communicate using standard Java naming APIs. The runtime client JNDI implementation is very lightweight and does not support federation of other name spaces, rather it provides a simple hierarchical name space for client applications. In most cases, applications leveraging JNDI do not need to interact directly with JNDI Name objects and simply use String representations of the names to be bound or located.

The JNDI provider does not persist objects or their state information across platform restarts, so the platform administrator is responsible for binding the objects each time the platform starts and configuring those objects as needed before binding them into the JNDI registry. While the application itself could programmatically register the objects that it needs each time the platform starts, the client platform provides another declarative model for JNDI bindings. Objects that need to be bound into JNDI can be declared using Eclipse extension points, so that when a lookup request is made for a specific object using its JNDI name, the JNDI provider will locate the declarative definition, create the object and return it to the client application on-demand. This "lazy" creation of objects provides for faster platform startup and memory allocation based on actual need, rather than expected need.

IBM

JNDI Provider

JNDI

JNDI object factories

- Used by JNDI Provider to create objects defined via Extension Points

- Three defined factories
  - ▶ com.ibm.pvc.jndi.provider.java.GenericObjectFactory
    - – creates objects of any type
    - – calls methods on objects
  - ▶ com.ibm.pvc.txncontainer.EJBObjectFactory
    - – creates EJB Objects
  - ▶ com.ibm.pvc.txncontainer.TxnDataSourceObjectFactory
    - – creates special type of DataSource for Transaction Container managed transactions
  - ▶ com.ibm.rcp.ws.objectfactory.WSObjectFactory
    - – provides a pre-initialized instance of Web services client stub for Apache Axis
  - ▶ com.ibm.msg.client.JMSConnectionFactory
    - – bind JMS administered objects to the Expeditor JNDI repository
  - ▶ com.ibm.rcp.database.core.ManagedDatasourceObjectFactory
    - – declare a managed datasource to the database infrastructure

Embedded transaction container and JNDI
© 2006 IBM Corporation

The JNDI object factories are used by JNDI provider to create objects defined using extension points. The three defined factories are listed on this slide.

JNDI Provider

JNDI

JNDI binding extension point

- com.ibm.pvc.jndi.provider.java.binding
  - ▸ defines an object that can be located via JNDI
  - ▸ references an object factory that knows how to create the object
  - ▸ Extension Registry searched on lookup request

```
<extension point="com.ibm.pvc.jndi.provider.java.binding">
  <binding
     jndi-name="java:comp/env/jdbc/dsname"
     objectFactory-id="com.ibm.pvc.jndi.provider.java.genericobjectfactory">
  </binding>
</extension>
```

JNDI Provider

JNDI

JNDI generic object extension point

- **com.ibm.pvc.jndi.provider.java.genericobject**
  - ▶ defines how to create object
    - object class
    - methods with parameters to call on object
  - ▶ used for general data source objects

```
<extension point="com.ibm.pvc.jndi.provider.java.genericobject">
  <object jndi-name="java:comp/env/jdbc/dsname"
         class="com.ibm.db2e.DB2eDataSource">
    <method name="setUrl">
      <method-parameter type="String" value="jdbc:db2e:oedb"/>
    </method>
  </object>
</extension>
```

Embedded transaction container and JNDI   © 2006 IBM Corporation

This slide describes the JNDI Generic Object Extension Point.

JNDI Provider

JNDI

JNDI object factory extension point

- **com.ibm.pvc.jndi.provider.java.objectfactory**
  - ▶ defines an object factory that can be used to create objects
  - ▶ new object factory needed
    - ▪ for specialized object type
    - ▪ genericobjectfactory does not suffice

```
<extension point="com.ibm.pvc.jndi.provider.java.objectfactory">
<objectfactory id="com.ibm.pvc.txncontainer.EJBObjectFactory"

class="com.ibm.pvc.txncontainer.EJBObjectFactory">
  </objectfactory>
</extension>
```

9   **Embedded transaction container and JNDI**   © 2006 IBM Corporation

While this capability is provided, it is not expected that you will create your own object factory instance.

IBM

Section

Embedded transaction container

Next, let's study the Embedded Transaction Container.

Transaction
Container
EJB 2.0 subset

Embedded transaction applications Enterprise Java Bean (EJB) 101

- Types of EJB's
  - ▶ Session – Performs a task for a client (for example, add or remove a book from a shopping cart)
  - ▶ Entity – Represents a business object (for example, order) that exists in persistent storage (for example, database)
  - ▶ Message-Driven – Acts as a listener for the JMS API, processing messages asynchronously
- State Management Modes for Session Beans
  - ▶ Stateful Session Beans – Maintains state for duration of client-bean session
  - ▶ Stateless Session Beans – Does not maintain state
- Persistence for Entity Beans
  - ▶ Bean-Managed Persistence (BMP) – Entity bean contains code to access persistent store (for example, database)
  - ▶ Container-Managed Persistence (CMP) – Enterprise Bean container automatically generates the necessary storage access calls based on bean's deployment descriptor

The Embedded Transaction Container (ETC) provides the ability to deploy Enterprise Java Beans to the client platform. Before we describe the details of the Embedded Transaction Container, here is a quick review of key concepts for Enterprise JavaBeans (or EJBs).

Transaction
Container
EJB 2.0 subset

Embedded transaction applications
Embedded transaction container (ETC)

- Supports the following subset of the EJB 2.0 specification:
  - ▸ Remote and Local Homes for local EJBs
  - ▸ Stateless Session Beans
  - ▸ Entity Beans, both BMP and CMP at both the EJB 1.1 and EJB 2.0 specification levels (local homes, use of abstract persistence schema)
  - ▸ Container-managed transactions
  - ▸ Entity bean tooling CMP support for container managed field types that implement java.io.Serializable.
  - ▸ CMP supports DB2e 8.2.1 and Cloudscape™ 10.0
  - ▸ JDBC Datasource support
  - ▸ JNDI support
  - ▸ Container-managed Relationships
- Toolkit compiles an EJB into a deployable Embedded Transaction Application bundle

Embedded transaction container and JNDI

The Embedded Transaction Container provides tooling and runtime support for local Enterprise Java Beans. The Embedded Transaction Container supports a subset of features in the EJB 2.0 specification as shown in this slide. The toolkit compiles and packages an EJB into a deployable bundle, called an Embedded Transaction Application bundle. The Embedded Transaction Application can be deployed onto any platform that is supported by the Embedded Transaction Container.

IBM

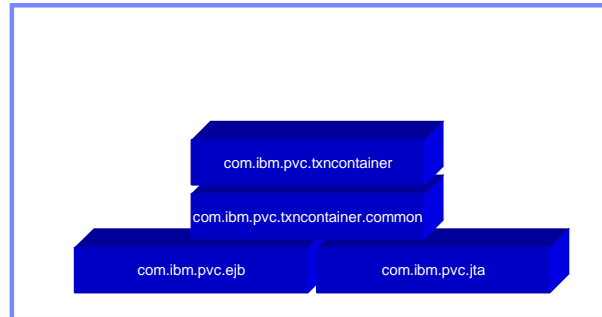Transaction
Container
EJB 2.0 subset

Embedded transaction applications ETC (cont.)

- Does <u>not</u> support the following features of the EJB 2.0 specification:
  - ▶ Stateful Sessions Beans
  - ▶ Pass-by-Copy semantics for mutable serializable objects when running in a single address space
  - ▶ For EJB 1.1, the Embedded Transaction Container does not persist references to an EJB's remote or remote home interfaces
    - This capability is not required for EJB 2.0
  - ▶ Message-driven Beans
  - ▶ Java Security support
  - ▶ EJB Query Language
  - ▶ Home methods
  - ▶ Bean-managed transactions

Embedded transaction container and JNDI

The Embedded Transaction Container is designed to be a light weight container, and, therefore, does not support certain features in the EJB 2.0 specification as shown in this slide. Please keep these restrictions in mind as you develop your Embedded Transaction Applications.

Transaction container infrastructure

Transaction
Container
EJB 2.0 subset

com.ibm.pvc.txncontainer

com.ibm.pvc.txncontainer.common

com.ibm.pvc.ejb   com.ibm.pvc.jta

Plug-in ID	Description
com.ibm.pvc.ejb	EJB APIs
com.ibm.pvc.jta	JTA APIs
com.ibm.pvc.txncontainer.common	Common classes shared between runtime and tools
com.ibm.pvc.txncontainer	Transaction Container Implementation

Embedded transaction container and JNDI

This slide shows the Transaction Container infrastructure on the client platform, including the plug-ins (components) installed with the client platform and a description of each of these plug-ins.

IBM

Transaction
Container
EJB 2.0 subset

Transaction container serviceability

- The Embedded Transaction Container leverages JSR 47 logging to log all messages

- JSR 47 logging is the Expeditor platform runtime logging framework
  - ▶ Default Settings
    - Logging turned on, default level is INFO
    - Tracing turned off

- Runtime Logging and Tracing levels can be modified by updating rcpinstall.properties

15   **Embedded transaction container and JNDI**   © 2006 IBM Corporation

The Embedded Transaction Container incorporates logging and tracing for serviceability. JSR 47 logging is used to log all messages. Runtime logging and tracing levels can be modified by updating rcpinstall.properties. Details can be found in the guide *Developing applications for Lotus Expeditor*.

IBM

Transaction
Container
EJB 2.0 subset

EJB packaging

- One or more beans exist in each bundle

- Definitions for JNDI binding generated by tools

- Bundle containing bean is started when bean lookup occurs

Embedded transaction container and JNDI

What is included in EJB packaging? One or more beans can exist in each bundle, definitions for JNDI binding generated by tools. The bundle containing the bean is started when bean lookup occurs.

IBM

Trademarks, copyrights, and disclaimers

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

Cloudscape   IBM   Lotus

EJB, J2SE, Java, JDBC, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information.   IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. 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 in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

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

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.

17   Embedded transaction container and JNDI   © 2006 IBM Corporation

This concludes the presentation.