User Documentation for J2EE CCI Resource Adapter and Gateway for Sun MTP

Table of Contents

1 OVERVIEW

The J2EE CCI ECI Resource Adapter for Sun™ Mainframe Transaction Processing (Sun MTP) (ECI Resource Adapter) is designed to allow programmatic access to data held inside Sun MTP from a Java™ client program. The Java program can be a standalone application, or an Enterprise Java Bean (EJB) running inside a J2EE container.

The programming model follows the Sun MTP External Call Interface model. That is, it provides the capability to remotely invoke a program running under Sun MTP and passing and returning parameters to and from that program using a Communications Area (COMMAREA).

When used as part of a standalone program, the resource adapter is said to be running in **unmanaged** mode

When used from inside a J2EE container, the resource adapter is said to be running in **managed** mode.

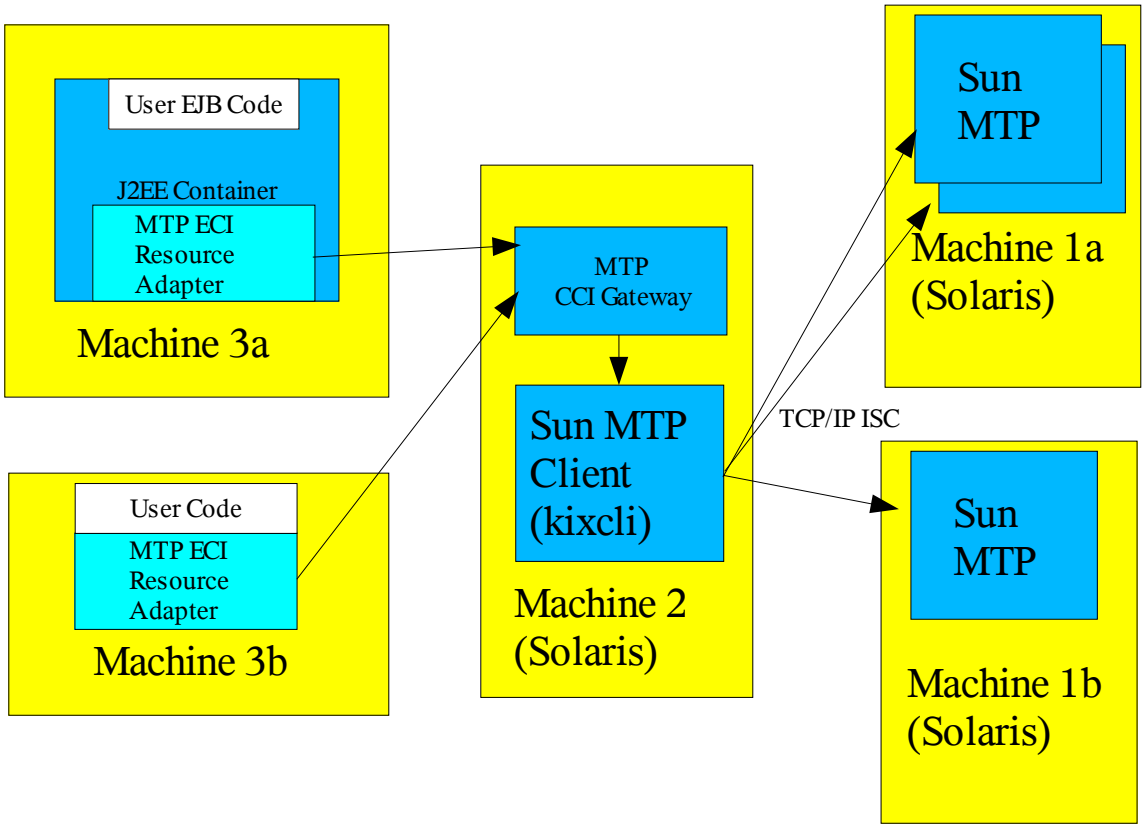
WARNING: At the time of this release, the support for resource adapters by most J2EE containers is at an early stage of provision, and is either not yet present, or is provided in a form that cannot yet be considered production quality. For these reasons, use of the Sun MTP ECI Resource Adapter in **managed** mode should be treated as an **early-access** version for **evaluation use only**, and is not currently supported in a production environment.

Notes about Transactions:

When using this version of the Sun MTP ECI Resource Adapter in **unmanaged** mode, transactional capabilites are available by invoking multiple programs in the same unit-of-work (local transaction in J2EE parlance). The Resource Adapter does **not** provide for integration with wider transaction mechanisms (global transaction in J2EE parlance).

When using this version of the Sun MTP ECI Resource Adapter in **managed** mode, transactional capabilities are **not** provided.

Communication from the Resource Adapter to one or more Sun MTP systems is achieved through a new gateway process (known as the Sun MTP CCI Gateway) which runs on top of Sun MTP Client for Solaris™. The resulting architecture is a three-tier architecture as illustrated in the following diagram.



In this diagram, several Sun MTP regions are shown running under Solaris on machines 1a and 1b.

The Sun MTP Client process (**kixcli**) is running on machine 2, along with a new gateway process known as the Sun MTP CCI gateway (**ccigateway**). The CCI Gateway process listens for requests made from remote machines by user code invoking the Sun MTP ECI Resource Adapter.

Machine 3b shows a piece of user-written code running in **unmanaged mode** (outside a J2EE container) and using the ECI Resource Adapter to contact the CCI Gateway and hence Sun MTP.

Machine 3a shows a piece of user-written code running in **managed mode** (as part of an EJB running inside a J2EE container) and using the ECI Resource Adapter to contact the CCI Gateway and hence Sun MTP.

The ECI Resource Adapter is written entirely in the Java programming language. When running in unmanaged mode, it can run in any environment that supports at least version 1.3 of Java 2 **Standard Edition** (J2SE v1.3).

When running in managed mode, the ECI Resource Adapter can be used in any J2EE container that supports version 1.3.1 of Java 2 **Enterprise Edition** (J2EE v 1.3.1). The ECI Resource Adapter is compliant with the J2EE Connector Architecture Specification version 1.0.

2 STARTING THE CCI GATEWAY

Before starting the CCI Gateway, the Sun MTP Client process (kixcli) must have been configured and started. For information about how to perform this, refer to the Sun MTP Client User's Guide.

After the kixcli process has been started, the CCI Gateway can be started. The CCI Gateway process is written mostly in Java, but makes native calls to access the Sun MTP Client. The CCI Gateway process can be run only on Solaris, and cannot be used with other versions of the Sun MTP Client (such as the 32-bit Microsoft Windows version).

The CCI Gateway process requires the use of Java 2 Standard Edition version 1.4 or later (J2SE 1.4).

Note that this differs from the version of Java required by the resource adapter, which only requires Java 2 Standard Edition version 1.3 or later (J2SE 1.3)

To start the CCI Gateway

1) Ensure that J2SE 1.4 (or later) is on your PATH. For example, if J2SE 1.4.2 is installed in /usr/j2sdk1.4.2, a Korn shell user might type: **export PATH=/usr/j2sdk1.4.2/bin:$PATH**

2) Ensure that your KIXCLI environment variable is set to match the value used when kixcli was started.

3) Run the **ccigateway** script located in the **bin** subdirectory of the installed Sun MTP Client software.

The CCI Gateway process starts with default parameters. Output similar to the following is displayed:

```
$ ccigateway
09/06/04 10:46: Creating name service on port 7865
09/06/04 10:46: Using name service at localhost:7865
09/06/04 10:46: Client timeout is 30 seconds
09/06/04 10:46: Binding system named "mtpsys01" into name service as "mtp/mtpsys01"
09/06/04 10:46: Binding system named "mtpsys02" into name service as "mtp/mtpsys02"
09/06/04 10:46: Startup is complete
```

The CCI Gateway process is an RMI-IIOP server process that publicizes its available services by means of a CORBA COS naming service. By default, the ccigateway script will start a new, in-process COS naming service on port 7865, through which the CCI Gateway's services will be publicized. The port number on which the new, in-process name services advertise its services can be changed by using the **-i** option on the ccigateway script. For example:

```
ccigateway -i 9999
```

Each system defined to the underlying **kixcli** process is advertised in the COS naming service. The name used is prefixed with **mtp/** to minimize the chances of clashes with other names in your COS naming service. The remaining part of the name is derived from the name of the underlying system as defined in the **kixcli.ini** file, but without trailing spaces.

If you want the systems to be advertised with a different bind prefix, specify the **-b** option. For example:

```
ccigateway -b myprefix/
```

If you want to use a pre-existing COS naming service located elsewhere within your network use the **-h** and **-p** options instead of the **-i** option. For example, if you already have a COS naming service running in your network on host **turtle** at port **9922**, you can start the ccigateway with the following command:

```
ccigateway -h turtle -p 9922
```

If the **-h** option is specified **without** the **-p** option, the port number used defaults to **7752**, which is the default port used by a Java-based COS naming service.

If the **-f** option is specified on startup of the ccigateway and the names of the systems being advertised already exist, this option forces the replacement of the old offers with the new ones. Without this option, the ccigateway will exit if it discovers that one of the names it is trying to register already exists in the COS namespace.

Specifying the **-r** option changes the timeout period for extended operations. When a client application executes SunMTP programs within a local transaction and then terminates without completing the transaction, the ccigateway must clean up these transactions. Transactions that have been waiting for client processing for longer than the timeout period will get terminated. This cleanup of such transactions is performed by the ccigateway periodically.

After the CCI Gateway is running, it is possible to run client code that uses the CCI Resource Adapter mode to make ECI-style calls to Sun MTP through the CCI Gateway.

Examples are provided with the product that illustrate the use of the resource adapter in both **managed** and **unmanaged** modes. Use of these examples is described in the following sections.

3 RUNNING THE EXAMPLES

3.1 The Unmanaged Mode Examples

Examples of unmanaged mode Sun MTP ECI Resource Adapter programs are provided in the **examples/java/unmanaged** subdirectory of the product installation in the files **Sample1.java** and **Sample2.java**.

Sample1.java demonstrates the programming model required to call SunMTP to initiate a SunMTP program passing it information and receiving a result. **Sample1.java** attempts to perform two ECI distributed program link (DPL) calls to the Sun MTP system through the ccigateway in order to retrieve the date and time from that system. If successful, the date and time are displayed.

Sample2.java is the same as **Sample1.java** except that it uses a JCA local transaction to wrapper the two requests. This demonstrates the use of local transaction facilities to get multiple SunMTP programs to execute within the same unit-of-work.

3.1.1 Prerequisites

- The example programs attempt to invoke the back-end Sun MTP program **ECIEX1S**, which is one of the example COBOL programs supplied with the Sun MTP Client product. Ensure that this program is correctly installed in your back-end Sun MTP system. Refer to the instructions in the Sun MTP Client User's Guide.
- Ensure that the kixcli process has been started on the middle tier.
- Ensure that the ccigateway process has been started on the middle tier.
- Ensure the J2SE 1.3 (or later) compiler and runtime environment are available.

3.1.2 Compiling and Executing the Unmanaged Mode Examples

A Makefile is provided to compile and run the example programs. The Makefile is intended for use on Solaris platforms. If you want to build and run the examples on another platform, the Makefile might need modification.

In the directory containing Sample1.java, type the following command to compile the Java source files:

make

Sample1.java can then be run by typing the following command:

make run1

Sample2.java can then be run by typing the following command:

make run2

Both examples have the same interface and provide the same output.

On starting, the example prompts you for the following information:

- The host name of the COS naming service used by the ccigateway process. For example, this might be **localhost** if you are attempting to run the sample on the same machine as the ccigateway, and you started the ccigateway with the default in-process COS naming service.
- The port number of the COS naming service used by the ccigateway process. For example, this might be **7865** if you started the ccigateway with the default in-process COS naming service.
- The name of the Sun MTP region that owns the ECIEX1S program, as defined to the Sun MTP Client. This should be the name of the region as displayed by the ccigateway when it was started, and **must include the bind prefix**. For example, in the example previously shown for starting the ccigateway, this might be **mtp/mtpsys01**.

The example program attempts to perform two ECI distributed program link (DPL) calls to the Sun MTP system through the ccigateway in order to retrieve the date and time from that system. If successful, the date and time are displayed.

Typical output is as follows:

```
Name Service Host (e.g. Localhost):
fliptop
Name Service Port (e.g. 7865):
7865
Region Name (e.g. mtp/mtpsys01):
mtp/mtpsys01
Date: 02/10/09
Time: 13:41:25
```

3.2 The Managed Mode Example

This section describes the steps necessary to build, deploy, and invoke the ECI Resource Adapter in **managed mode**, that is, within a J2EE compliant container that supports the Java Connector Architecture (JCA) version 1.0.

At the time of this release, only early access versions of production-level J2EE containers were available that supported JCA 1.0, and a number of these containers exhibited significant problems with their JCA 1.0 support.

For this reason, the following cautionary notes apply:

- Use of the Sun MTP ECI Resource Adapter in these containers should be considered as an early access version for evaluation use **only**.
- Instructions for the use of the Sun MTP ECI Resource Adapter in managed mode is provided only for the Java 2 Enterprise Edition Reference Implementation version 1.3.1 (J2EE RI 1.3.1).
- The Sun MTP ECI Resource Adapter running in managed mode is **not** supported in a production environement.
- The Sun MTP ECI Resource Adapter running in managed mode does **not** provide any transactional capabilities.

3.2.1 Running the Managed Mode Example in Java 2 Enterprise Edition version 1.3.1

3.2.1.1 Prerequisites

- You must have access to, and be familiar with, the J2EE version 1.3.1 Reference Implementation. Specifically, you must be familiar with the starting and stopping of J2EE, and with the use of the deployment tool **deploytool**. Additionally, you must be familiar with the concepts of Enterprise Java Beans and their deployment and invocation.
- This managed mode example consists of a **stateless session bean** that attempts to invoke the back-end Sun MTP program **ECIEX1S**, which is one of the example COBOL programs supplied with the Sun MTP Client product. Ensure that this program is correctly installed in your back-end Sun MTP system. Refer to the instructions in the Sun MTP Client User's Guide.
- Ensure that the **kixcli** process has been started on the middle tier.
- Ensure that the **ccigateway** process has been started on the middle tier.
- Ensure that your **J2EE** system is using **J2SE** version1.3 or later and that both the java compiler (**javac**) and runtime environment are available.

3.2.1.2 Overview of Procedure

Use of the managed mode example consists of the following steps:

1. Deploying the Sun MTP ECI Resource Adapter to your J2EE RI system.
2. Building Managed Sample 1.
3. Deploying Managed Sample 1 to your J2EE RI system.
4. Invoking the client program that uses Managed Sample 1.

These steps are described in the following sections.

3.2.1.3 Deploying the Sun MTP ECI Resource Adapter to your J2EE System

For use in managed mode, the Sun MTP ECI Resource Adapter is supplied as a J2EE Resource Archive (RAR file) named **mtpeci.rar**, which is located in the **lib** directory of the Sun MTP Client product installation. It is recommended that this resource archive is independently deployed to your J2EE RI system. That is, it is deployed separately from the Enterprise Java Beans (EJBs) that will invoke it.

Deployment of **mtpeci.rar** is performed using the J2EE RI **deploytool**.

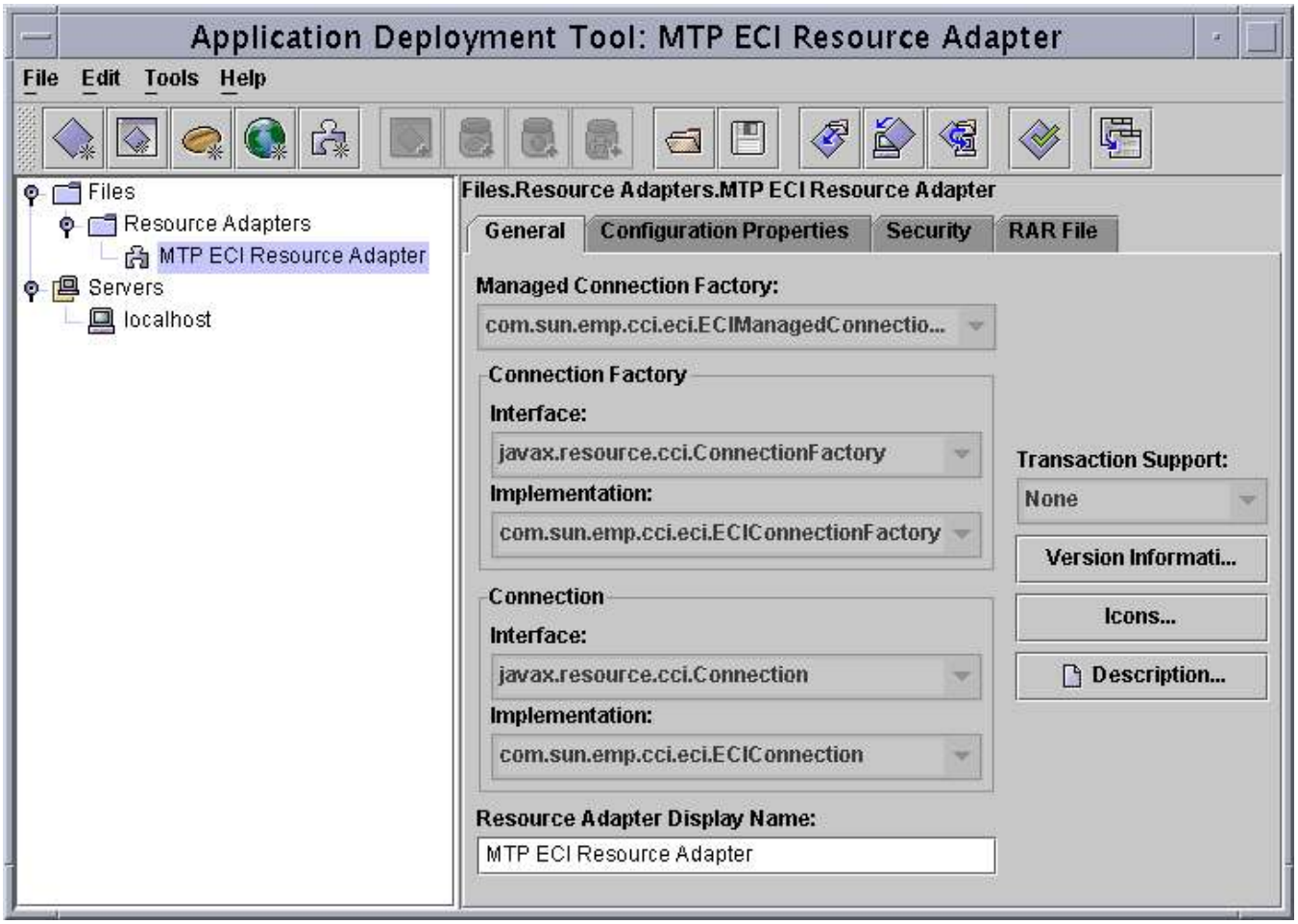
To deploy mtpeci.rar

1. Ensure that your J2EE RI system is running.

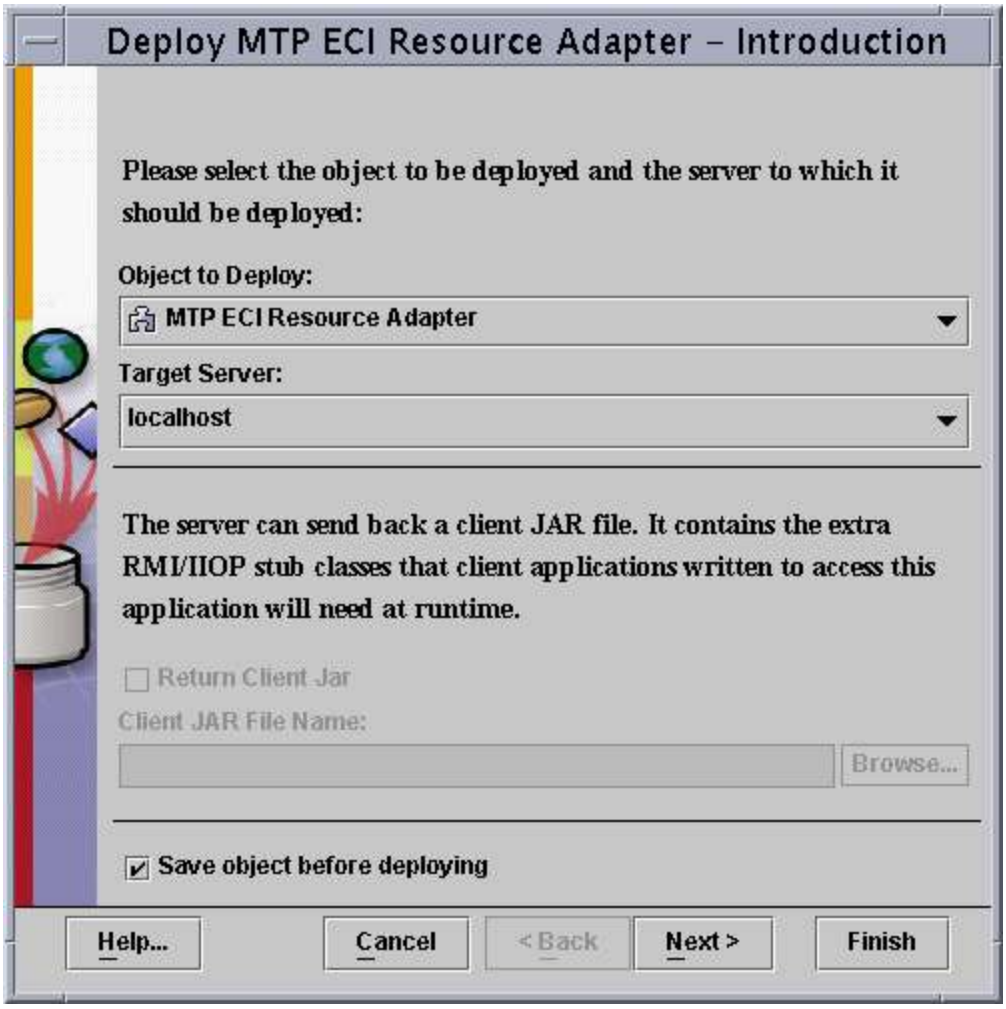
2. Start **deploytool**.

3. In **deploytool**, choose **'File...open...'** and select **mtpeci.rar** as the file to open.

deploytool displays a window similar to that shown below:



4. Select the **MTP ECI Resource Adapter** entry in the left pane of the deploytool display with the right mouse button, and select **Deploy...** from the pop-up menu. A dialog is displayed:

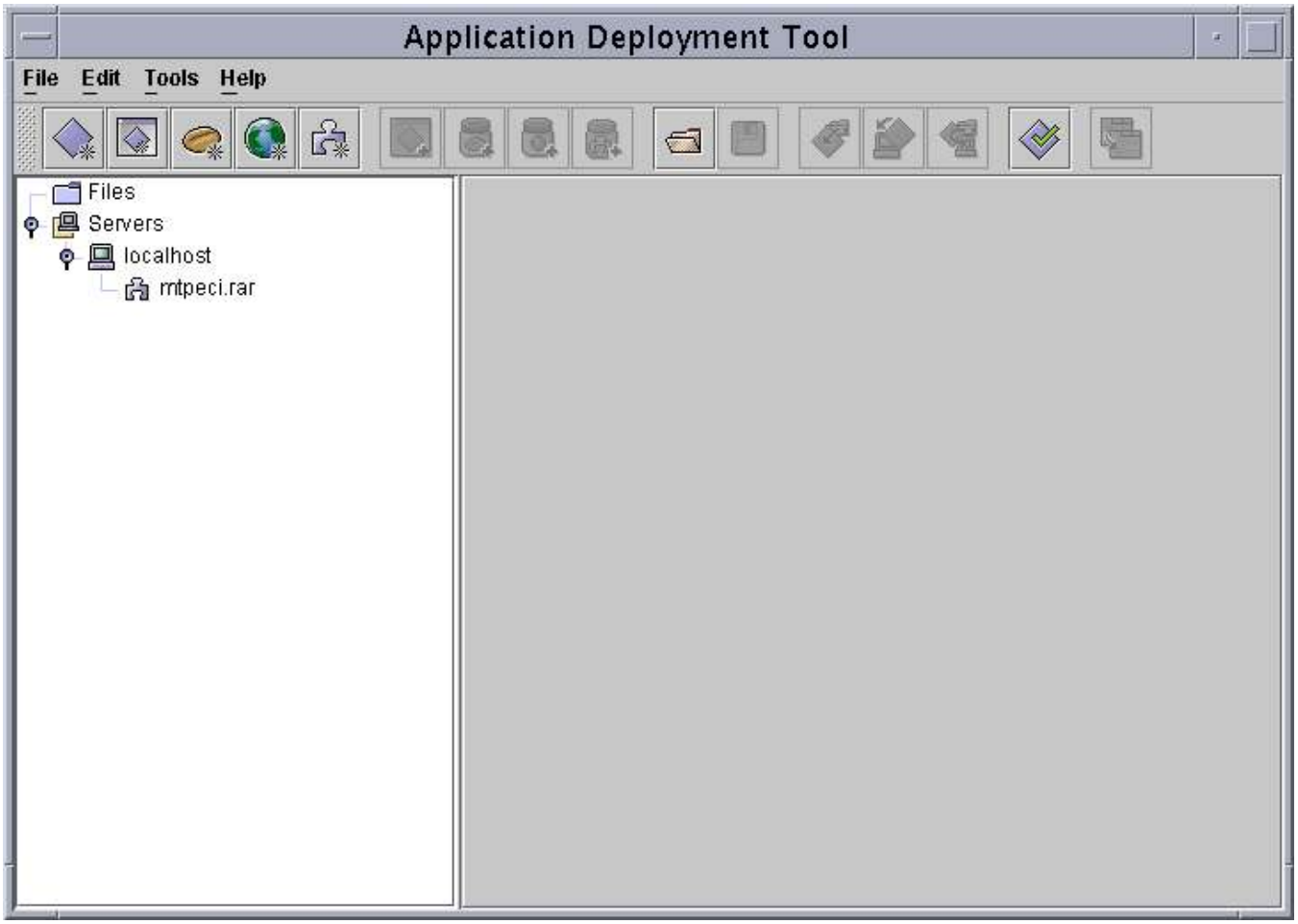


5. Ensure that the correct **Target Server** is selected and press **Finish**.

6. Close the **MTP ECI Resource Adapter** entry in deploytool.

Warning: Failure to do this can create problems in deploytool later in the process.

The **deploytool** display changes to the following:

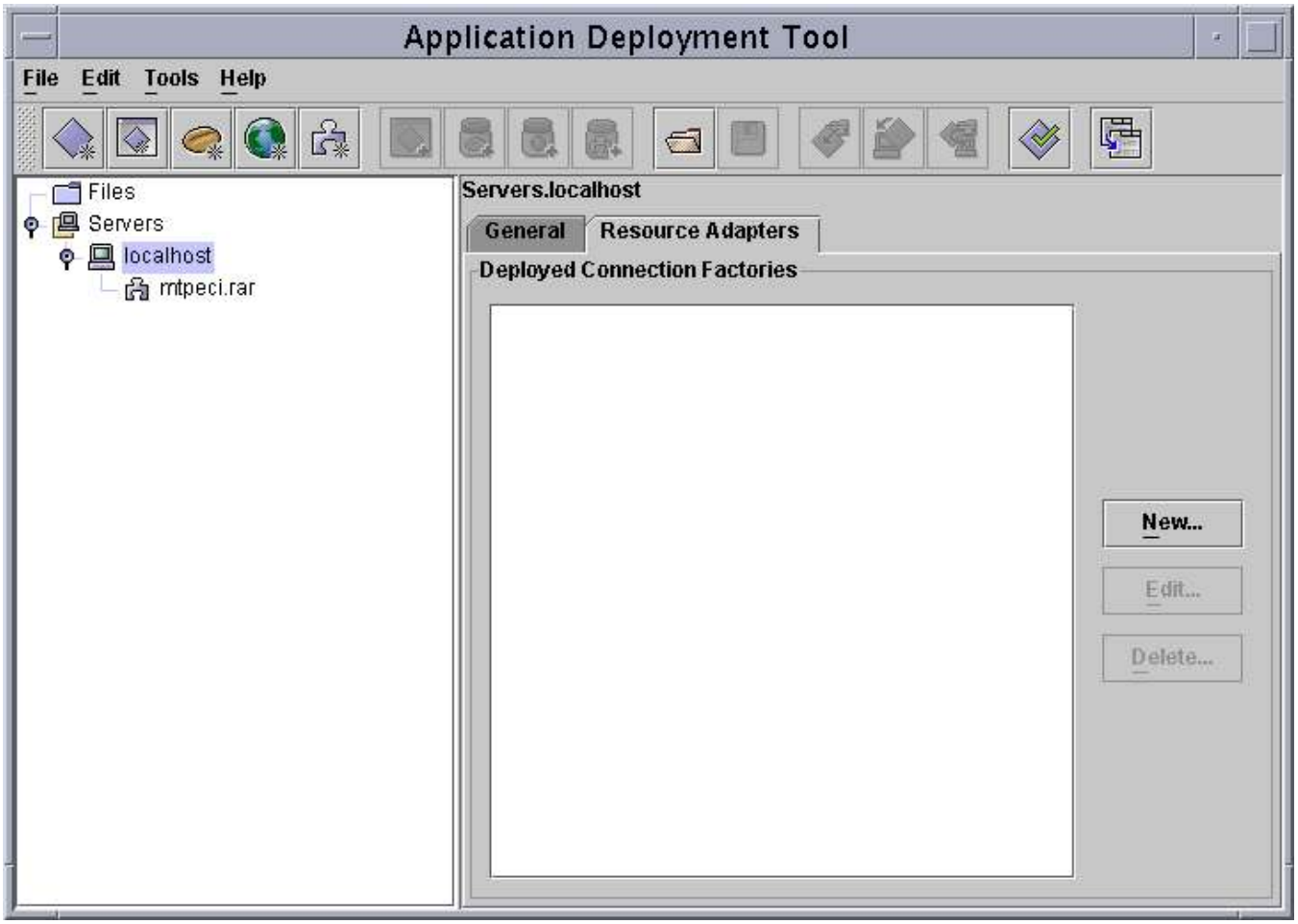


At this point, the ECI Resource Adapter is deployed, but no **connection factory** instances have been defined. A connection factory instance must be defined for each Sun MTP system that you want to access using the ECI Resource Adapter. A connection factory instance has a **JNDI** name by which that instance is known, and a number of **properties** by which the correct Sun MTP system is identified.

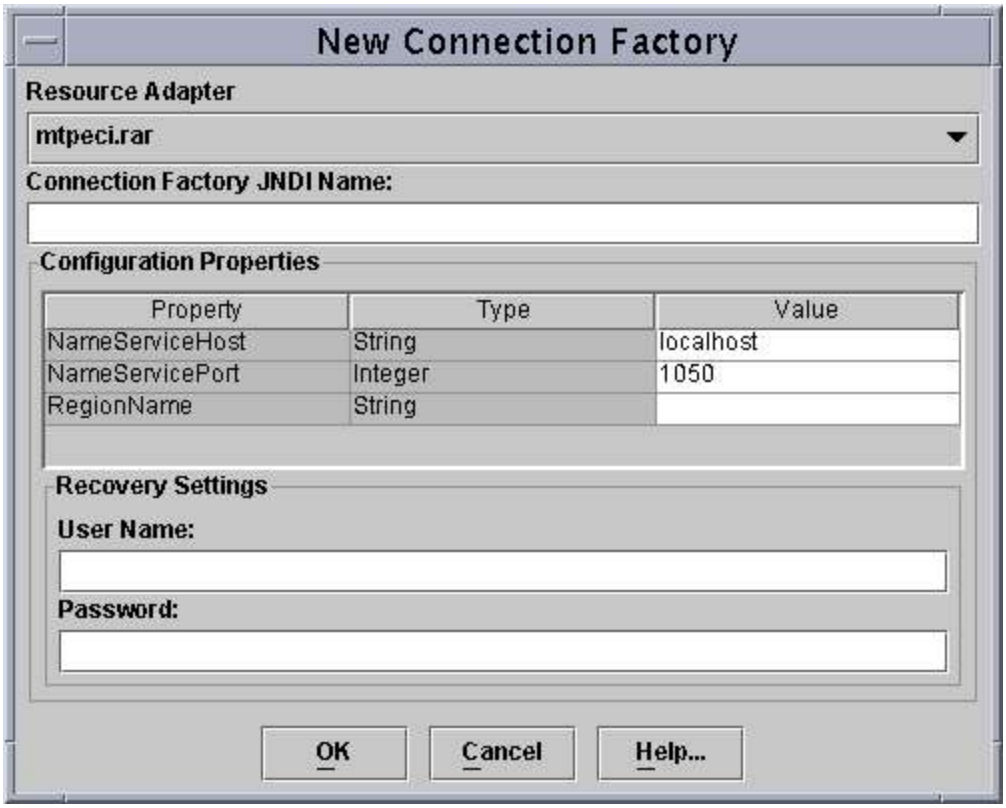
To define a **connection factory** instance

1. Select the relevant server in the pane on the left hand side of the deploytool display.

2. Select the **Resource Adapters** tab in the display on the right hand pane of deploytool. The following window is displayed:



3. Click the **New...** button. The following dialog is displayed:



4. In the **Connection Factory JNDI Name** field, type the name by which this connection factory is to be known to the J2EE RI container. You can prefix the name with **eis/** to indicate an Enterprise Information System. This name will be the name of the factory used to create connections to a Sun MTP system. For consistency with the names used in the example of starting the **ccigateway**, the name **eis/mtpsys01**is used.

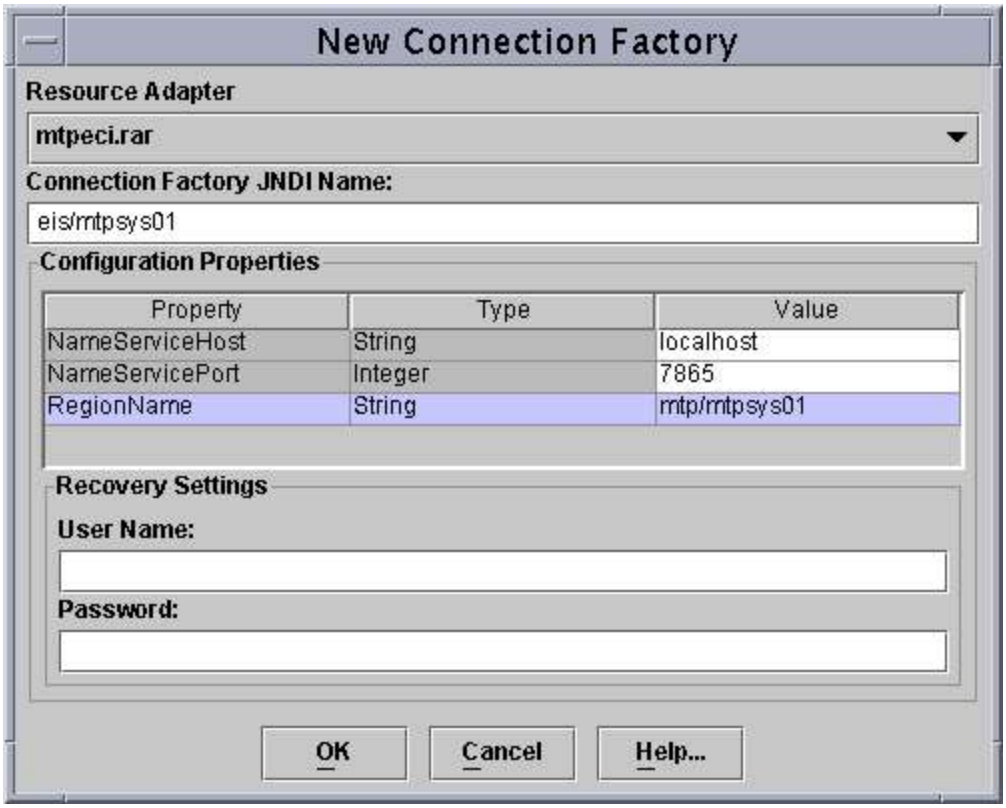
5. The three configuration properties **NameServiceHost**, **NameServicePort**, and **RegionName** must be specified. These properties are defined as follows:

NameServiceHost: The host on which the COS name service used by the ccigateway is running. This is printed out when the ccigateway starts. If the J2EE RI system is running on the same host as this COS name service, 'localhost' can be specified. Otherwise, provide the fully qualified host name.

NameServicePort: The port number of the COS name service used by the ccigateway. This is printed out when the ccigateway starts.

RegionName: The name of the relevant Sun MTP region as defined in the COS name service by the ccigateway, including any bind prefix.

If the ccigateway was started with the parameters used for illustration earlier in this document, and the J2EE RI system is running on the same host as the ccigateway which is using an in-process COS name service, the values shown in the following dialog can be set:



Type the values appropriate for your environment and click the **OK** button.

Warning: If you return at a later date to edit these values, the values you specified are not displayed. The default values are shown instead. This is a limitation in J2EE RI 1.3.1.

At this point, it might be helpful to confirm that the resource adapter has been successfully deployed. To do this, stop your J2EE RI system and restart it with the **-verbose** option. During startup a message similar to the following is displayed:

```
Binding Connection Factory, name = eis/mtpsys01
```

The resource adapter is now successfully deployed and a connection factory successfully defined.

3.2.1.4 Building Managed Sample 1

The source of an example stateless session bean that uses the Sun MTP ECI Resource Adapter in managed mode is located in the **examples/java/managed** subdirectory of the Sun MTP Client product installation. This is referred to as **Managed Sample 1**.

A Makefile is provided to build the source files into a deployable J2EE **EAR** file. This Makefile is intended for use on the Solaris operating system and might require modification if the sample EJB is to be built on any other platform.

Managed Sample 1 is built from a number of Java source files, XML files, and other data files. The main source code is in file **MS1Bean.java**. This file contains the EJB code to call the back-end Sun MTP system through the CCI Resource Adapter. You might find it instructive to examine this source code.

To build the stateless session bean

1. Ensure that the **J2EE_HOME** environment variable is set to point to the location of your J2EE RI installation.

2. Type **make**.

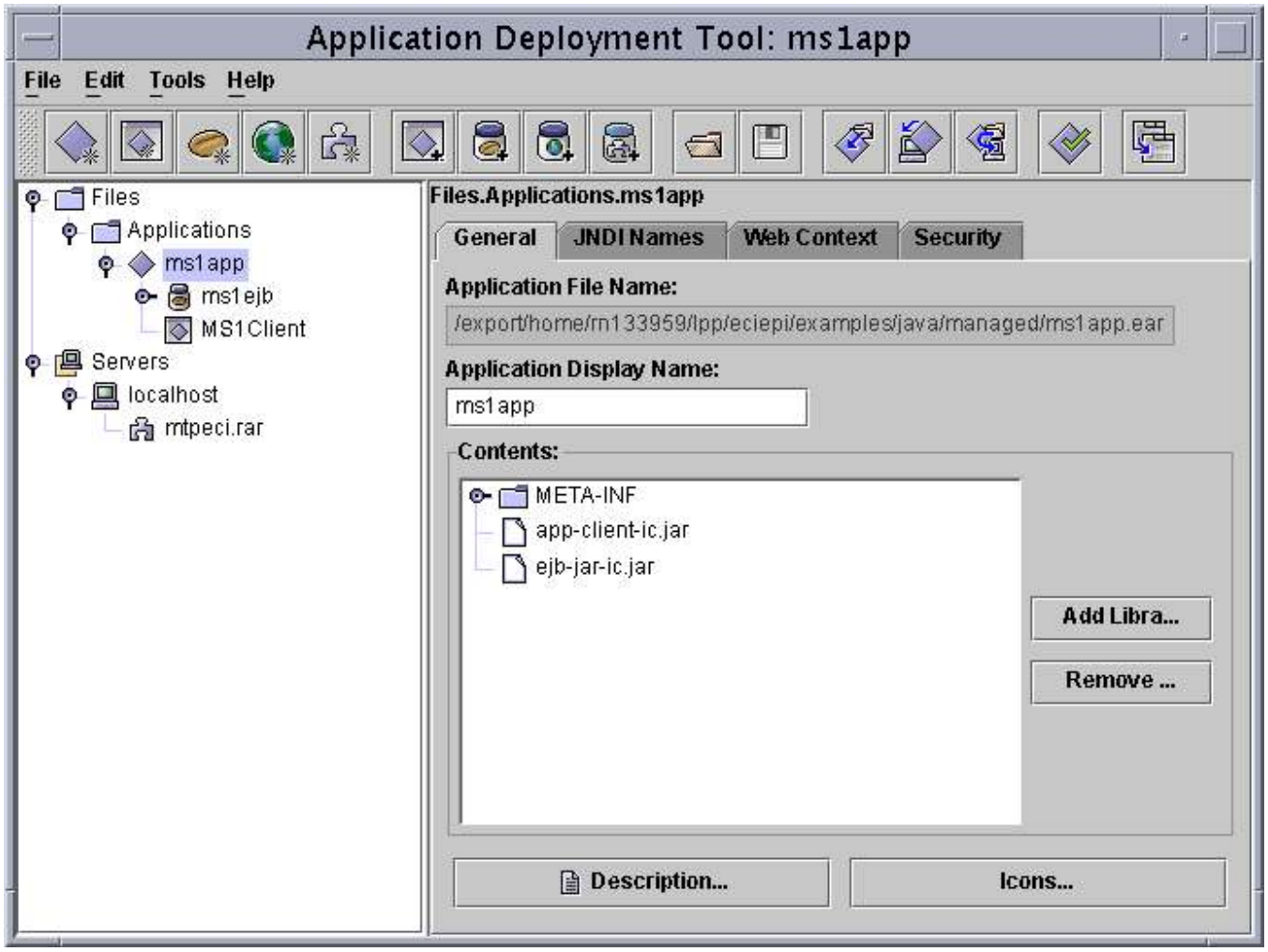
This should result in the generation of a file called **ms1app.ear**. This is the EJB application containing Managed Sample 1, which must now be deployed.

3.2.1.5 Deploying Managed Sample1 to your J2EE RI System

To deploy Managed Sample1

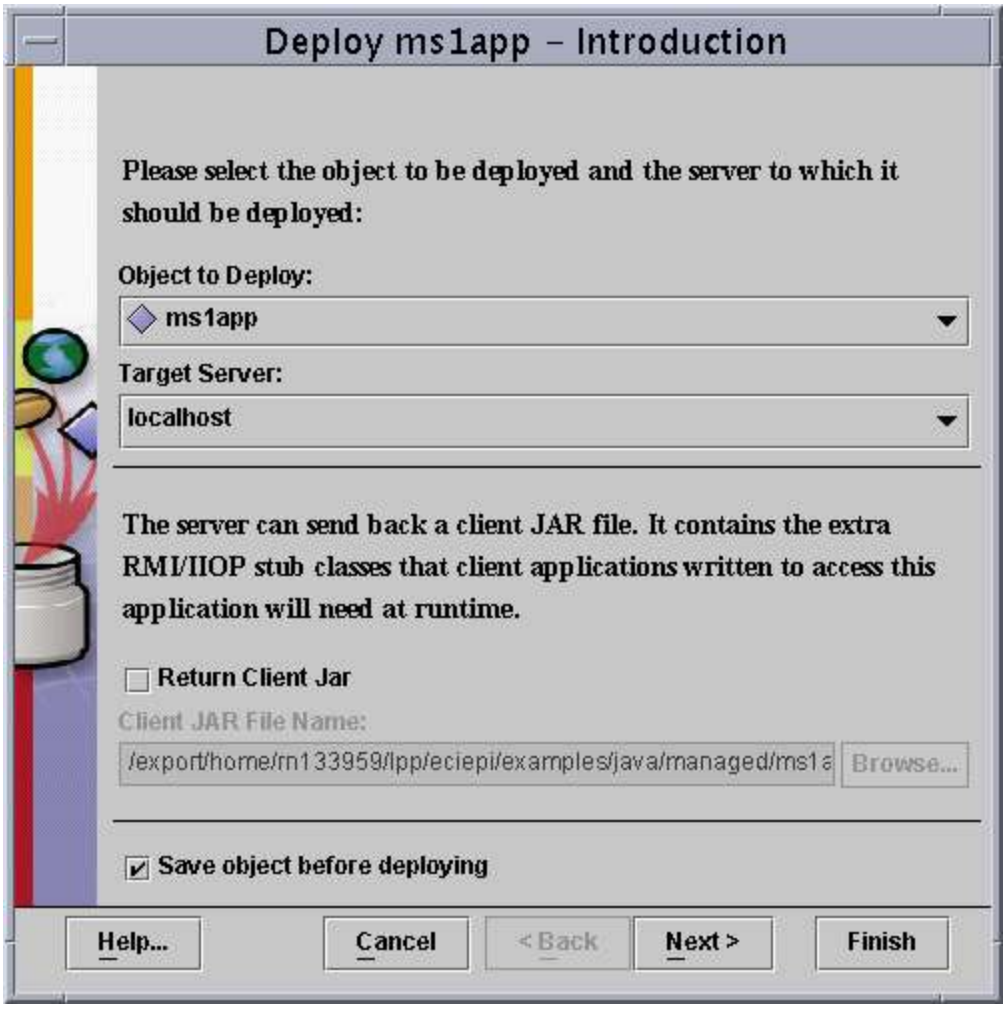
1. In **deploytool**, choose '**File...open...'** and select **ms1app.ear**.

Deploytool shows a display similar to the following:



2. Right-click **ms1app** in the left pane of **deploytool** and select **Deploy...** from the pop-up menu.

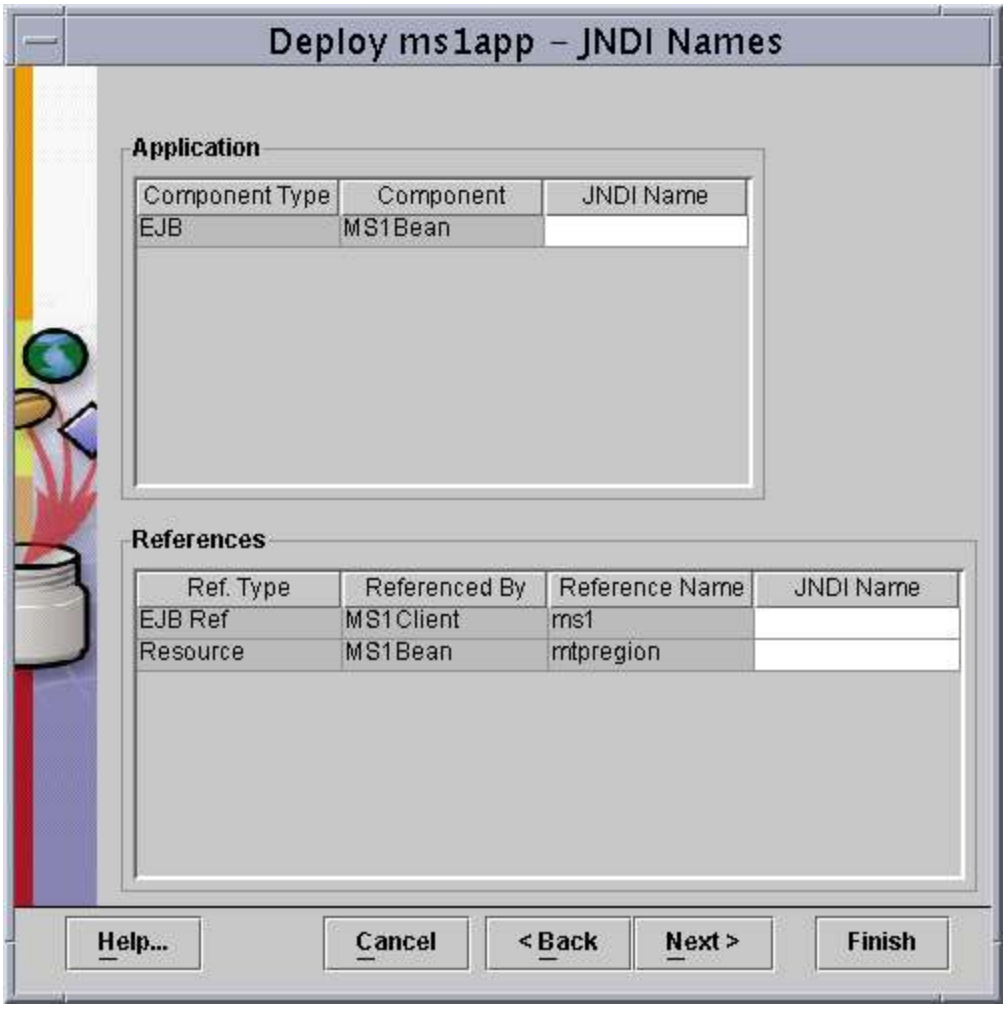
The following dialog is displayed:



3. Check the **Return Client Jar** check box and ensure that the **Client JAR File Name** is set to the directory containing Managed Sample 1.

4. press **Next >.**

The following dialog is displayed:



5. You must provide three pieces of information:

- The JNDI Name with which this EJB is going to be identified to the J2EE RI container. You can use the value **ejb/ms1** here.
- The JNDI Name which the EJB **Client** code will use to locate the EJB. This must be the same as the value in 1, so use the value **ejb/ms1** here also.
- The JNDI name of the Sun MTP ECI connection factory as known to the J2EE RI Container. In our example this is **eis/mtpsys01**.

6. Then press **Next >,** and press **Finish** on the next page of the dialog. The EJB is deployed, and the client JAR returned.

The EJB application is now successfully deployed. At this point it is beneficial to stop and restart the J2EE RI system with the **-verbose** flag. Lines similar to the following are displayed during startup:

```
Binding name:`java:comp/env/mtpregion`
Loading jar:/opt/j2ee131/repository/fred/applications/ms1app1035819551642Server.jar
```

Note: These messages might not appear in the order shown above.

3.2.1.6 Invoking the Client Program That Uses Managed Sample 1

During the above processes the client program **MS1Client.java** was built and made available for use. It can be invoked using the **run** target in the supplied **Makefile**.

To invoke the client program

1. Ensure that the **bin** directory of your J2EE RI installation (for example, **$J2EE_HOME/bin**) is on your **PATH**.

2. Type **make run**.

3. Press **Return** when prompted for a Username and Password, as this example does not use security.

The client program is then invoked. The client program attempts to call the EJB, and thus the MTP system, five times to obtain the current time. For more details, refer to the source code of **MS1Client.java**. If all is well, output similar to the following is displayed:

```
/home/eciepi/examples/java/managed make run
APPCPATH=ms1appClient.jar runclient -client ms1app.ear -name MS1Client -textauth
Initiating login ...
Enter Username:
Enter Password:
Binding name:`java:comp/env/ms1`
MS1Client: lookup got IOR:
000000000000025524d493a6d616e616765642e4d5331486f6d653a303030303030303030303030303
030300000000000000001000000000000000ec000102000000000f3132392e3135362e3133392e323600
00821f00000000003eafabcb00000000242bb119b1000000010000000000000010000000d5472616e7
369656e74504f4100000000000000000d2ba1e0f900000000000000001ff0a000000000050000000010000
002000000000000010001000000020501000100010020000101090000000100010100000000210000000
800000000000000000000000190000002d000000000000000025687474703a2f2f666c6970746f703a3931
39312f6d7331617070436c69656e742e6a617200000000000000001f000000040000000300000020000000
00400000001
0 : 15:49:19
1 : 15:49:19
2 : 15:49:19
3 : 15:49:19
4 : 15:49:19
```

The last five lines of output are the time as returned from Sun MTP. Diagnostic and informational messages are displayed in the window from which the J2EE RI System was started.

4 API OVERVIEW

The API implemented by the Sun MTP ECI Resource Adapter is described in detail in Javadoc documentation supplied with the product. To view this Javadoc, use a web browser and open the file **index.html** supplied in the **doc/api** subdirectory of the product installation.

A brief overview of the API and its current limitations is included here.

4.1 Introduction

The Java Connector Architecture (JCA) provides for access to Enterprise Information Systems. The Sun MTP ECI Resource Adapter conforms to this architecture and provides ECI access to Sun MTP systems. The paradigm used by code that invokes JCA compliant resource adapters is as follows:

1) Obtain access to a **javax.resource.cci.ConnectionFactory** for the relevant EIS.

2) Obtain a **javax.resource.cci.Connection** to that EIS from the **ConnectionFactory**.

3) Perform a **javax.resource.cci.Interaction** with the EIS via the **Connection**.

4) Release the **Connection**.

When running in **managed** or **unmanaged** modes, the details of how these steps are performed vary, and are described separately.

Note that this initial version of the Sun MTP ECI Resource Adapter does **not** make any provision for extended units-of-work. That is, local transaction and global transaction in J2EE parlance are **not** supported.

4.1.1 Unmanaged Mode API Overview

In unmanaged mode, a **javax.resource.cci.ConnectionFactory** instance is obtained by first instantiating a **com.sun.emp.cci.eci.ECIManagedConnectionFactory**, setting its properties to refer to the relevant Sun MTP system, and then calling its **createConnectionFactory()** method.

This is typically achieved by a piece of code similar to the following:

```
ECIManagedConnectionFactory emcf = new ECIManagedConnectionFactory();
```

```
emcf.setNameServiceHost("turtle");
emcf.setNameServicePort(7865);
emcf.setRegionName("mtp/mtpsys01");

ConnectionFactory cf = (ConnectionFactory)emcf.createConnectionFactory();
```

After a ConnectionFactory has been obtained, a connection to the associated Sun MTP system can be obtained by calling the **getConnection()** method in the **ConnectionFactory**.

```
Connection c = cf.getConnection();
```

If it is required to execute multiple Sun MTP programs within a single unit-of-work, or if explicit control over the commit and rollback operation is required, then a local transaction must be started. This is done using the following:

```
c.getLocalTransaction().begin();
```

After the **Connection** has been obtained, and a transaction started (if necessary), an **Interaction** is required. An **Interaction** provides a way to call the Sun MTP system and is obtained from a **Connection** using the **createInteraction()** method:

```
Connection c = cf.getConnection();
Interaction ia = c.createInteraction();
```

The **Interaction** can then be used to execute a call to Sun MTP. For this, an **InteractionSpec** must be supplied, as well as input and output **Record**s. Typical code is as follows:

```
ECIInteractionSpec eis = new ECIInteractionSpec();
eis.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE);
eis.setFunctionName("ECIEX1S");
```

SYNC_SEND_RECEIVE is currently the only interaction verb supported by the Sun MTP ECI Resource Adapter, and this means that a synchronous call will be made to the Sun MTP system with the results available as soon as the call returns.

The **Function Name** is the name of the program to invoke on the remote Sun MTP system. (This is the name of the program as defined in the region's Processing Program Table (PPT)).

A record containing the **COMMAREA** to send on the call to Sun MTP can be supplied as follows:

```
GenericRecord gr = new GenericRecord("ASKTIME:".getBytes());
```

(Where GenericRecord is com.sun.emp.cci.eci.GenericRecord)

The call can then be made to Sun MTP by invoking the **execute()** method of the **Interaction**:

```
ia.execute(eis, gr, gr);
```

In this case we have chosen for the return COMMAREA to be stored in the same Record as held in the initial COMMAREA.

We can execute any number of Sun MTP programs at this point. If a local transaction has been started, then all executions will execute within that extended unit-of-work. Otherwise, each program will be its own unit-of-work.

If we have chosen to run our Sun MTP program within an extended unit-of-work, then we must explicitly commit or roll back that unit of work. This is done using either

```
c.getLocalTransaction().commit();
```

or

```
c.getLocalTransaction().rollback();
```

Note: If a transaction was not explicitly started, then this operation is not required.

All that remains now is that when the program has finished using the **Connection**, it should be released by calling its **close()** method:

```
c.close();
```

Note: If a local transaction is left in flight at the point that the connection is closed, then the transaction will be backed out.

For more information about the use of the ECI Resource Adapter in unmanaged mode, see the example described in section 3.1.

4.1.2 Managed Mode API Overview

The primary difference in the use of the API in managed mode is that the **ConnectionFactory** is obtained by an EJB from its container by means of a JNDI lookup, rather than by explicit creation. This allows the J2EE container to implement optimizations, such as connection pooling.

Typically, in the managed environment, a **ConnectionFactory** is obtained by an EJB as follows:

```
Context ic = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)ic.lookup("java:comp/env/mtpregion");
```

The attributes of the **ConnectionFactory** (the properties describing which Sun MTP system it will access, and its JNDI name) are specified during deployment of the ECI Resource Adapter **RAR** file and during the deployment of the EJBs.

After a **ConnectionFactory** has been obtained, the programming paradigm is effectively the same as for the unmanaged case, and is not described here.

For more information about the use of the ECI Resource Adapter in managed mode, see the example described in section 3.2.