

TIBCO® Enterprise Administrator Developer's Guide

*Software Release 1.0.0
November 2013*

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO and Two-Second Advantage are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 2013 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

TIBCO Documentation and Support Services	4
TIBCO® Enterprise Administrator Concepts	5
TIBCO Enterprise Administrator SDK Architecture	5
Components of TIBCO Enterprise Administrator	6
Configuring SSL between the Server and the Agent	8
Setting up TIBCO Enterprise Administrator Agent Library	9
Running TIBCO Enterprise Administrator Agent Library in the Server mode	9
Running TIBCO Enterprise Administrator Agent Library in the Servlet mode	10
Agent ID	11
Starting the Sample TIBCO Enterprise Administrator Agent	12
Defining TIBCO Enterprise Administrator Object Types	14
Defining TIBCO Enterprise Administrator Object Types using Interfaces	14
Defining a Top Level Object Type	15
Defining an Object Type	16
Defining an Object Type with Only One Instance	18
Object Types Optional Aspects	19
Defining TIBCO Enterprise Administrator Object Types Using Annotations	19
Concepts	21
Defining Operation for Objects	22
Asynchronous Operations Support	24
Defining References for Object Types with Interfaces	26
Defining References for Object Types with Annotations	27
Object ID	28
Solution	29
Roles	30
Permissions	32
User Interface Customization	34
Configuring TIBCO Enterprise Administrator Agent for Auto-Registration	35

TIBCO Documentation and Support Services

All TIBCO documentation is available in the TIBCO Documentation Library, which can be found here:
<http://docs.tibco.com>

Product-Specific Documentation

The following documents can be found in the TIBCO Documentation Library for TIBCO Enterprise Administrator:

- *TIBCO® Enterprise Administrator User's Guide*
- *TIBCO® Enterprise Administrator Installation*

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support as follows:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

<http://www.tibco.com/services/support>

- If you already have a valid maintenance or support contract, visit this site:

<https://support.tibco.com>

Entry to this site requires a user name and password. If you do not have a user name, you can request one.

How to Join TIBCOCommunity

TIBCOCommunity is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCOCommunity offers forums, blogs, and access to a variety of resources. To register, go to:

<http://www.tibcommunity.com>

TIBCO® Enterprise Administrator Concepts

TIBCO Enterprise Administrator provides a centralized administrative interface to manage and monitor multiple TIBCO products deployed in an enterprise.

You can perform common administrative tasks such as authenticating and configuring runtime artefacts across all TIBCO products within one administrative interface. You can also manage products that do not have a complete administrative interface, providing you a unified and simplified administrative experience.

The following are the salient features of TIBCO Enterprise Administrator:

- **Centralized Administration:** TIBCO Enterprise Administrator provides a single-point access to multiple products deployed across an enterprise. You can easily manage and monitor runtime artifacts.
- **Simple to use:** TIBCO Enterprise Administrator is simple to install, develop, use, and maintain.
- **Shared Services Model:** TIBCO Enterprise Administrator shares common administrative concepts across all products thereby promoting a consistent and reusable shared services model.
- **Pluggable and Extensible:** As your enterprise evolves, you can add new products to the TIBCO Enterprise Administrator.
- **Rich set of API:** TIBCO Enterprise Administrator Agent Library allows organizations to develop custom TIBCO Enterprise Administrator agents to manage TIBCO and non-TIBCO products and applications. TIBCO products such as TIBCO BusinessWorks™ Express 1.0.0 and TIBCO Collaborative Information Manager™ 8.3.1 provide agents for TIBCO Enterprise Administrator. You can develop your own agents to expose your product on TIBCO Enterprise Administrator. The product comes with a set of API that is both declarative and extensible. You can develop your own agents and decide what part of your product needs to be rendered on TIBCO Enterprise Administrator.
- **Support for Interactive Shell:** TIBCO Enterprise Administrator provides a command-line utility called TIBCO Enterprise Administrator Shell. You can use the shell to perform almost all the tasks offered by the Web-based GUI.

TIBCO Enterprise Administrator SDK Architecture

TIBCO Enterprise Administrator is based on an agent-based architecture. TIBCO Enterprise Administrator SDK comes with two main components: the TIBCO Enterprise Administrator server and the agent library.

The TIBCO Enterprise Administrator provides two distinct user interfaces: a Web-based GUI and a command-line based shell interface. A product being managed using the TIBCO Enterprise Administrator must have a product agent registered with the TIBCO Enterprise Administrator server. You can develop your own agents for your products using the agent library. The TIBCO Enterprise Administrator SDK ships with samples that demonstrate the various aspects of developing product agents. To see your product on TIBCO Enterprise Administrator, perform the following steps:

1. Develop an agent.
2. Compile and start the agent.
3. Register the agent with the TIBCO Enterprise Administrator server.

Develop an agent

An agent represents your product in TIBCO Enterprise Administrator. An agent identifies the assets on the product that need to be rendered on the TIBCO Enterprise Administrator. Use the agent library to model the set of assets available in your product. The agent library provides you with five basic concepts to represent your product on TIBCO Enterprise Administrator. The concepts are: Process,

Application, Resource, Access_Point, and Top_Level. For example, in the ActiveMatrix world, a node is an asset of an operating system Process, the DAA file is An application, an environment is a Group, an Enterprise is a Top_Level concept, and SOAP is the Access_Point. In this manner, assets available in your product can be modelled into a concept provided by the agent library.

Every asset has an attribute, an action, and a relationship associated with it. For example, some *attributes* of an ActiveMatrix node are the name of a node, the default state, and the location of the node. Creating a node, starting or stopping a node are the *actions* performed on the node. The correlation that the node has with its environment is the *relationship* it shares with the environment. As an agent developer, you must start by identifying the assets that need to be modelled using the agent library. You must define the attributes, actions, and relationships of each asset.

Compile and Start the Agent

After developing the agent, compile and start the agent by running the appropriate ant scripts. The steps are mentioned in [Starting the Sample TIBCO Enterprise Administrator Agent](#).

Register an Agent with the Server

After starting the agent, register the agent with the TIBCO Enterprise Administrator server. The steps are listed in [Step 6](#).

Components of TIBCO Enterprise Administrator

The TIBCO Enterprise Administrator comprises a server, an agent corresponding to a product, a server UI and the shell interface.

The TIBCO Enterprise Administrator has the following components:

The Server

The server is the equivalent of a Web Server. The server is hosted within a Web server and caters to the HTTP requests coming from the browser. The server manages the communication between the browser and agents. The server interacts with the agent to get data about the products registered on the TIBCO Enterprise Administrator. The server is responsible for:

- Collecting data on all the products registered with it
- Maintaining a cache of the data; thereby promoting faster searches
- Hosting all the TIBCO Enterprise Administrator server views
- Auto-registering an agent with itself
- Providing details about the machines on which the products are running
- Providing user management features such as granting and revoking a user's permissions

The Agent

An agent is a bridge between the TIBCO Enterprise Administrator server and a product. When an agent is registered with the TIBCO Enterprise Administrator, it discovers the product that must be exposed to the administrator. The agent creates a graph of objects specific to the product that needs to be rendered on the TIBCO Enterprise Administrator server UI. The agent interacts with the server using the REST API. TIBCO Enterprise Administrator agents can run in any of the following ways: standalone, embedded, or hosted. TIBCO Enterprise Administrator comes with an extensible set of API that helps you develop your own agents for your products. An agent provides the following basic concepts:

- Group: is a container of artefacts. For example, a cluster, domain, and ActiveMatrix environment.
- Process: is any operating system process. For example, a BusinessWorks engine, and ActiveMatrix node.

- **Resource:** is a shareable configuration or artefact. For example, a JMS connection, and port number.
- **Application:** is any deployable archive. For example, a WAR, and DAA.
- **Access_Point:** is the result of a deployment. For example, a service point, and queue.

To develop a customized agent, you can either extend these concepts or use annotations.

Server UI

TIBCO Enterprise Administrator provides a default UI to manage and monitor products. You can customize labels and icons on the UI to match the object types of your product. You can add more views to suit your product requirements.

Shell

TIBCO Enterprise Administrator provides a command-line utility called the TIBCO Enterprise Administrator shell. It is a remote shell based on the SSH protocol. The Shell is accessible using any terminal program such as, Putty. The scripting language is similar to bash from UNIX. You can use the Shell to perform almost all the tasks offered by the server UI.

Configuring SSL between the Server and the Agent

To enable SSL, when you start the agent, you should set the SSL system properties.

Procedure

- Set the following SSL properties to enable SSL between the server and the agent:

Property	Description
tea.agent.http.keystore	The file or URL of the SSL Key store location
tea.agent.http.keystore.password	The password for the key store
tea.agent.http.cert.alias	Alias of SSL certificate
tea.agent.http.keymanager.password	The password for the specific key within the key store
tea.agent.http.truststore	The file name or URL of the trust store location
tea.agent.http.truststore.password	The password for the trust store.
tea.agent.http.idletimeout	The maxIdleTime to set, which translates to the Socket.setSoTimeout(int) call.
tea.agent.http.threadpool.acceptors	The number of acceptor threads to set.

Setting up TIBCO Enterprise Administrator Agent Library

You can configure and setup the TIBCO Enterprise Administrator Agent library either in the server or the servlet modes.

Running TIBCO Enterprise Administrator Agent Library in the Server mode

In the server mode, the TIBCO Enterprise Administrator Agent runs as a standalone process. The TIBCO Enterprise Administrator Agent Library comes bundled with the Jetty server which will be used for serving the agent service endpoints.

Procedure

- To configure the TIBCO Enterprise Administrator Agent library to run in server mode, instantiate an object of the class `com.tibco.tea.agent.server.TeaAgentServer`. There are a few overloaded constructors available to instantiate the `TeaAgentServer`. The one used in this example takes the following arguments:

Option	Description
name	Name of the agent
version	Version of the agent
agentinfo	Description for the agent
hostname	Hostname for the jetty connector
port	Port for the jetty connector
context-path	Path for ServletContext
enable-metrics	Enables metrics for the TIBCO Enterprise Administrator SDK Agent Library

```
TeaAgentServer server = new TeaAgentServer("HelloWorldAgent", "1.0", "Hello
World Agent", 1234, "/helloworldagent", true);
```

- To register Object Types with the TIBCO Enterprise Administrator Agent library, use any of the following: `com.tibco.tea.agent.server.TeaAgentServer.registerInstance()` and `com.tibco.tea.agent.server.TeaAgentServer.registerInstances()`. The `registerInstance()` method takes an instance of a `TeaAgent` as a parameter. The `registerInstances()` method takes a `varargs` parameter that receives a variable number of Object Types.

```
server.registerInstance(new HelloWorldAgent());
server.registerInstances(arg0);
```
- You can configure the TIBCO Enterprise Administrator Agent library to customize the content specific to your requirements. The content includes HTML, CSS, javascript, images, and so on. Use `com.tibco.tea.agent.server.TeaAgentServer.registerResourceLocation()` to register the resource location that has these files.

```
server.registerResourceLocation(file);
```
- (Optional) To disable the default search capability of an agent registered in the server, use the method `disableIndex()` on the server instance.

```
server.disableIndex();
```
- After the TIBCO Enterprise Administrator agent server has been configured, use `com.tibco.tea.agent.server.TeaAgentServer.start()` to initiate and start the TIBCO Enterprise Administrator agent server.

```
server.start();
```

Starting the sample: HelloWorldAgent

```
TeaAgentServer server = new TeaAgentServer("HelloWorldAgent", "1.0", "Hello
World Agent", 1234, "/helloworldagent", true);
server.registerInstance(new HelloWorldAgent());
server.registerInstances(arg0)
server.registerResourceLocation(file);
server.start();
```

Running TIBCO Enterprise Administrator Agent Library in the Servlet mode

TIBCO Enterprise Administrator provides an abstract servlet that needs be subclassed to register object instances.

Procedure

1. Extend the abstract servlet of TIBCO Enterprise Administrator to define the object that needs to be registered.

The following example code defines only one object to register in the server.

```
public class HelloWorldServlet extends TeaAgentServlet {

    /*
     * (non-Javadoc)
     * @see com.tibco.tea.agent.server.TeaAgentServlet#getObjectInstances()
     */
    @Override
    protected Object[] getObjectInstances() throws ServletException {
        return new Object[]{new HelloWorldAgent()};
    }
}
```

2. Configure the servlet with proper parameters using web.xml. Map the agent servlet to /* as further dispatches are done by the servlet.

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4">

    <display-name>HelloWorld Agent</display-name>

    <servlet>
        <servlet-name>HelloWorldAgent</servlet-name>
        <servlet-class>HelloWorldServlet</servlet-class>
        <init-param>
            <param-name>name</param-name>
            <param-value>HelloWorldAgent</param-value>
        </init-param>
        <init-param>
            <param-name>version</param-name>
            <param-value>1.0</param-value>
        </init-param>
        <init-param>
            <param-name>agent-info</param-name>
            <param-value>HelloWorld Agent</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloWorldAgent</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>

</web-app>
```

3. Deploy the servlet using the standard servlet container mechanisms. To verify, you can start an embedded Jetty server, with the programmatically deployed servlet. For a standalone agent process, configuration through the server mode is a better approach.

```
public static void main(final String[] args) throws Exception {

    Server server = new Server(1234);
    ServletContextHandler servletContextHandler = new
ServletContextHandler(server, "/helloworldagent", false, false);

    ServletHolder servletHolder =
servletContextHandler.addServlet(HelloWorldServlet.class, "/*");
    servletHolder.setInitParameter("name", "HelloWorldAgent");
    servletHolder.setInitParameter("version", "1.0");
    servletHolder.setInitParameter("agent-info", "HelloWorld Agent");
    server.start();

    server.join();
}
```

Agent ID

By default, TIBCO Enterprise Administrator uses the name provided during the registration of the agent to uniquely identify an agent. You can override that behavior by providing an agent identifier that will be used instead of the name.

Registering another agent with the same id will automatically unregister the previously registered agent. Ensure that the Agent identifier is set before agent server is started. For example:

```
server.setAgentId("uniqueAgentId");
server.start();
```

Starting the Sample TIBCO Enterprise Administrator Agent

TIBCO Enterprise Administrator SDK ships with samples that demonstrate the various aspects of developing product agents. This procedure lists how to start the sample, `HelloWorldAgent.java` that is located under `TIBCO_HOME/tea/1.0/samples/helloworld`.

Procedure

1. Navigate to `TIBCO_HOME/tea/1.0/samples/helloworld`.
2. Open `TIBCO_HOME/tea/1.0/samples/helloworld/helloworld/src/com/tibco/tea/agent/HelloWorldAgent.java`. By default, the following code is displayed. You can make your changes to the code to suit your needs.

```
import com.tibco.tea.agent.TeaAgent;
import com.tibco.tea.agent.TeaAgentServer;
import com.tibco.tea.agent.api.TeaObjectType;
import com.tibco.tea.agent.api.TeaConcept;
import com.tibco.tea.agent.api.TeaGetInfo;
import com.tibco.tea.agent.api.TeaOperation;
import com.tibco.tea.agent.api.TeaParam;
import com.tibco.tea.agent.types.AgentObjectInfo;
import java.io.IOException;

@TeaObjectType(name = "HelloWorldAgent", concept = TeaConcept.TOP_LEVEL,
description = "Hello World Agent")

public class HelloWorldAgent {
    @TeaGetInfo
    public AgentObjectInfo getInfo() {
        AgentObjectInfo agentObject = new AgentObjectInfo();
        agentObject.setName("helloworld");
        agentObject.setDesc("Hellow World Agent");
        return agentObject;
    }
    @TeaOperation(name = "helloworld", description = "Send greetings")
    public String helloworld(
        @TeaParam(name = "greetings", description = "Greetins parameter")final
String greetings)
        throws IOException {
        return "Hello " + greetings;
    }
    public static void main(final String[] args) throws Exception {
        TeaAgent agent = new TeaAgent("HelloWorldAgent", "1.0", "Hellow World
Agent");
        agent.registerInstance(new HelloWorldAgent());
        TeaAgentServer server = new TeaAgentServer(agent, 1234, "/"
helloworldagent");
        server.start();
    }
}
```

3. Open the *TIBCO_HOME/tea/1.0/samples/helloworld/build.xml*. By default, the following code is displayed. You can make your changes to the code to suit your needs.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="agent" default="compile">
  <target name="compile">
    <javac srcdir="src" destdir="dist">
      <classpath>
        <fileset dir="../../lib">
          <include name="*.jar"/>
        </fileset>
      </classpath>
    </javac>
  </target>

  <target name="start">
    <java classname="HelloWorldAgent" fork="yes">
      <classpath>
        <pathelement location="dist"/>
        <fileset dir="../../lib">
          <include name="*.jar"/>
        </fileset>
      </classpath>
    </java>
  </target>
</project>
```

4. From the command prompt, navigate to *TIBCO_HOME/tea/1.0/samples/helloworld*. To build the java file, run `ant`.
5. To start the agent, run `ant start`.
6. To register the agent with the TIBCO Enterprise Administrator, perform the following steps:
 - a) Login to TIBCO Enterprise Administrator.
 - b) From the Agents panel, click **Register**.
 - c) In the Register Agent window, enter the Agent Name as `helloworld`.
 - d) Enter the Agent URL as `http://localhost:1234/helloworldagent`.
 - e) Click **Register**.

The Agent pane is displayed and the helloworld agent appears as a registered agent.
7. To access the agent, from the Agent pane, click **HelloWorldTopLevelType**.

Result

You can access the agent registered on the server.

Defining TIBCO Enterprise Administrator Object Types

TIBCO Enterprise Administrator Object Types are defined either by extending the interfaces or marking the classes with annotations.

Defining TIBCO Enterprise Administrator Object Types using Interfaces

You can define various Object Types by implementing interfaces provided as a part of the TIBCO Enterprise Administrator agent library.

There are three interfaces which can be used for defining object types:

- `com.tibco.tea.agent.api.TopLevelTeaObject`
- `com.tibco.tea.agent.api.TeaObject`
- `com.tibco.tea.agent.api.SingletonTeaObject`

You also can assign some aspects to these object types to retrieve additional information.

Defining a Top Level Object Type

Procedure

1. A top level object type can be defined by creating a class that implements `com.tibco.tea.agent.api.TopLevelTeaObject`

For example,

```
public class MyProduct implements TopLevelTeaObject {

    public String getTypeName() {
        //..
    }

    public String getTypeDescription() {
        //..
    }

    public String getName() {
        //..
    }

    public String getDescription() {
        //..
    }

    public Collection(BaseTeaObject) getMembers() {
        //..
    }

    @TeaOperation(description = "get", methodType = MethodType.READ)
    public Result get() {
        //..
    }
}
```

Code explanation:

- To define an object type which represents your product, create a class which implements `com.tibco.tea.agent.api.TopLevelTeaObject`.
 - To describe the top level object type, use `com.tibco.tea.agent.api.TopLevelTeaObject.getTypeName()` and `com.tibco.tea.agent.api.TopLevelTeaObject.getTypeDescription()`
 - As there is only one instance possible of this type, the instance specific methods are merged with object type specific methods. `com.tibco.tea.agent.api.TopLevelTeaObject.getName()` and `com.tibco.tea.agent.api.TopLevelTeaObject.getDescription()` should be used for describing the single instance of the top level object type.
 - `com.tibco.tea.agent.api.TopLevelTeaObject.getMembers()` should be used to return the members of the single instance of top level object type.
 - Any `TeaOperation` should be defined using the `TeaOperation` annotation.
2. Register the top level object type definition with the TIBCO Enterprise Administrator agent library.

For example, create an instance of the top level object type and register it with the `TeaAgentServer`.

```
TeaAgentServer server = //..
// register other instances including the top level object type
MyProduct myProduct = //..
server.registerInstance(myProduct);
server.start();
```

Defining an Object Type

Any object type other than the top level object can be defined using the `com.tibco.tea.agent.api.TeaObject`. The instances of these object types are represented by the object instantiated by the class implementing `com.tibco.tea.agent.api.TeaObject`.

Procedure

1. To define an object type using the `com.tibco.tea.agent.api.TeaObject`, provide a class which implements `com.tibco.tea.agent.api.TeaObjectProvider`.

For example,

```
public class MyObjectTypeProvider implements TeaObjectProvider<MyObjectType> {

    public String getTypeName() {
        //..
    }

    public String getTypeDescription() {
        //..
    }

    public TeaConcept getConcept() {
        //..
    }

    public MyObjectType getInstance(final String key) {
        //..
    }
}
```

code explanation:

- Create a class which implements `com.tibco.tea.agent.api.TeaObjectProvider`.
- `com.tibco.tea.agent.api.TeaObjectProvider.getTypeName()` and `com.tibco.tea.agent.api.TeaObjectProvider.getTypeDescription()` should be used for describing the object type.
- `com.tibco.tea.agent.api.TeaObjectProvider.getConcept()` should be used for defining the concept type for this object type. The various concept types are explained later in the document.
- `com.tibco.tea.agent.api.TeaObjectProvider.getInstance()` should be used for key to instance mapping. When the TIBCO Enterprise Administrator agent library needs to obtain an instance corresponding to a given key for an object type, it will depend on the `getInstance()` method. For example, when an operation needs to be invoked on a particular instance of an object type, the TIBCO Enterprise Administrator agent library depends on this method for getting the instance by passing the key as a parameter.

2. Define the object type by implementing the `com.tibco.tea.agent.api.TeaObject`.

For example,

```
public class MyObjectType implements TeaObject {

    public String getName() {
        //..
    }

    public String getDescription() {
        //..
    }

    public String getKey() {
        //..
    }

    @TeaOperation(description = "get", methodType = MethodType.READ)
    public Result get() {
        //..
    }
}
```

code explanation:

- Create a class which implements `com.tibco.tea.agent.api.TeaObject`.
 - `com.tibco.tea.agent.api.TeaObject.getName()` and `com.tibco.tea.agent.api.TeaObject.getDescription()` should be used for describing the instance of the object type.
 - `com.tibco.tea.agent.api.TeaObject.getKey()` should be used for returning the key of the instance of the object type.
 - Any `TeaOperation` should be defined using the `TeaOperation` annotation.
3. Register the object type definition with the TIBCO Enterprise Administrator Agent library.

For example, create an instance of the object type provider that you created and register that with the `TeaAgentServer`.

```
TeaAgentServer server = //..
// register other instances including the top level object type
MyObjectTypeProvider myObjectTypeProvider = //..
server.registerInstance(myObjectTypeProvider);
server.start();
```

Defining an Object Type with Only One Instance

Procedure

1. To define an object type which only has a single instance, create a class that implements `com.tibco.tea.agent.api.SingletonTeaObject`.

For example,

```
public class MySingleton implements SingletonTeaObject {

    public String getTypeName() {
        //..
    }

    public String getTypeDescription() {
        //..
    }

    public String getName() {
        //..
    }

    public String getDescription() {
        //..
    }

    public TeaConcept getConcept() {
        //..
    }

    @TeaOperation(description = "get", methodType = MethodType.READ)
    public Result get() {
        //..
    }
}
```

Code explanation:

- To define an object type which only has a single instance, create a class which implements `com.tibco.tea.agent.api.SingletonTeaObject`.
- `com.tibco.tea.agent.api.SingletonTeaObject.getTypeName()` and `com.tibco.tea.agent.api.SingletonTeaObject.getTypeDescription()` should be used for describing the singleton object type.
- Since there is only one instance possible for this type, the instance specific methods are merged with object type specific methods. `com.tibco.tea.agent.api.SingletonTeaObject.getName()` and `com.tibco.tea.agent.api.SingletonTeaObject.getDescription()` should be used for describing the single instance of the singleton object type.
- `com.tibco.tea.agent.api.SingletonTeaObject.getConcept()` should be used to return the type of concept of the object type.
- Any `TeaOperation` should be defined using the `TeaOperation` annotation(explained later in the developer's guide).

2. Register the singleton object type definition with the TIBCO Enterprise Administrator Agent library.

For example, create an instance of the singleton object type and register it with the TeaAgentServer.

```
TeaAgentServer server = //..
// register other instances including the top level object type
MySingleton mySingleton = //..
server.registerInstance(mySingleton);
server.start();
```

Object Types Optional Aspects

The TIBCO Enterprise Administrator Agent library supports optional aspects for defining the object type: members, status, and configuration.

To control the definition of an object type, the following interfaces can be added to the interfaces that are used to create object types.

- com.tibco.tea.agent.api.WithMembers
- com.tibco.tea.agent.api.WithStatus
- com.tibco.tea.agent.api.WithConfig

Define an object type with members

An object type can have members or subclasses that the object type contains or refers to. For example, an enterprise might have environments, a machine might have processes, a domain might have applications and so on. Such relationships can be defined using com.tibco.tea.agent.api.WithMembers. TopLevelTeaObject is expected to support this interface.

Define an object type with status

An object type can have a status. For example, a process might be running or stopped. Such object types can be defined using the com.tibco.tea.agent.api.WithStatus.

Define an object type with configuration

An object type can have a configuration. For example, an environment might have virtualization layer configuration or the node might have port configuration. These can be defined using the com.tibco.tea.agent.api.WithConfig.

Defining TIBCO Enterprise Administrator Object Types Using Annotations

User defined classes can be annotated to define TIBCO Enterprise Administrator Object Types. One Java class can be used to define one or more TIBCO Enterprise Administrator Object Types.

The main annotation that defines the TEA Object Type is @TeaObjectType. While defining it in the java class, specify the name of the type, concept, and description. The object type should at least have a method with @TeaGetInfo annotation. When a key is passed to the method, it should return basic information about the object. An exception to this rule is the TOP_LEVEL type. Since there is only one instance of the TOP_LEVEL type per agent, the getInfo() method does not take a key as an argument. You can define a *key* that helps you identify the instances of an object type. A key is a String that is not processed by the server and not visible on the UI.

Procedure

1. To define TIBCO Enterprise Administrator Object Type, mark the java class with the `@TeaObjectType` annotation. Specify the name of the type, concept, and description. You can use the `@TeaObjectTypes` annotation to define multiple types on the same class. The `getInfo()` method is mandatory for all types, except for top level objects where the method does not take a key as an argument. You can also have domain-specific implementation of obtaining the actual object. For example:

```
@TeaObjectType(name="TOMCAT_SERVER", concept=TeaConcept.PROCESS,
description="Tomcat Server")
public class TomcatServerManager {

    @TeaGetInfo
    public AgentObjectInfo getInfo(String key) {

        Server server = lookupTomcatServer(key);

        AgentObjectInfo result = new AgentObjectInfo();
        result.setName(server.getName());
        result.setDesc(server.getDescription());
        return result;
    }

    Server lookupTomcatServer(String key) {
        // domain specific implementation
    }
}
```

2. For an Object Type with many types on the same java class, the minimal implementation is that there is a method marked with the `@TeaGetInfo` annotation. The annotation on the methods has the type they belong to using the `objectType` attribute. This attribute is mandatory when more than one type is defined on a class. Mark the method that takes a key as a parameter and returns basic information about an object, with the `@TeaGetInfo` annotation.

For example,

```
@TeaObjectTypes({
    @TeaObjectType(name = "TOMCAT_SERVER", concept = TeaConcept.PROCESS,
description = "Tomcat Server"),
    @TeaObjectType(name = "TOMCAT_WEBAPP", concept = TeaConcept.APPLICATION,
description = "Tomcat Web Application") })
public class TomcatServerManager {

    @TeaGetInfo(objectType = "TOMCAT_SERVER")
    AgentObjectInfo getServerInfo(String key) {
        // ...
    }

    @TeaGetInfo(objectType = "TOMCAT_WEBAPP")
    AgentObjectInfo getWebappInfo(String key) {
        // ...
    }
}
```

3. (Optional) Mark the method with the `@TeaGetConfig` annotation to retrieve the configuration. *Configuration* is a Java Bean object that can be displayed on the TIBCO Enterprise Administrator server.

For example,

```
@TeaGetConfig
MyJavaBean getConfiguration(String key) {
    // ...
}
```

4. (Optional) Mark the method with the `@TeaGetStatus` annotation to retrieve the current state of the object.

State is a custom label that is associated with an object. The label can be defined by the agent and it should represent a state in the life-cycle of the object. Some operations can be linked to particular states of the object. For example, you should not start an application that is already running.

The `getStatus()` method returns the `AgentObjectStatus` instance with the state name and description.

```
@TeaGetStatus
AgentObjectStatus getStatus(String key) {
    // sample code, it should depend on the state of actual object
    AgentObjectStatus result = new AgentObjectStatus();
    result.setState("DEPLOYED");
    result.setDesc("Application is deployed successfully");
    return result;
}
```

5. (Optional) Mark the method with the `@TeaGetMembers` annotation to retrieve the members list. The `getMembers()` method returns an array of `AgentObjectIdentifier` instances.

```
@TeaGetMembers
AgentObjectIdentifier[] getMembers(String key) {
    // ...
}
```

Concepts

There are several concepts defined in TIBCO Enterprise Administrator. Each object type has to be marked as a concept type. The concepts do not have a different behaviour or default functionality. However, they have a different default icon. They define basic interoperability between products.

Following are the different types of concepts:

PROCESS

Is a type that represents an operating system process or a similar concept. A process should have state and configuration. It can have subprocesses.

APPLICATION

Is a type that represents an application running inside the process or a similar concept. An application should have state and configuration. It can have subprocesses.

RESOURCE

Is a type that represents a shared object or configuration. It should have configuration; however having a state is optional.

ACCESS_POINT

Is an endpoint like SOAP or REST. It should have both, configuration and state.

TOP_LEVEL

Is a special type that represents the root-level object in the tree. There can be only one instance of the object per agent. It cannot have a configuration or state. Note that unlike other concepts, methods that access objects on this type do not need the key argument.

Defining Operation for Objects

A method on one of the registered object type can be exposed as an operation available to TIBCO Enterprise Administrator server and the user interface.

Procedure

1. Define a method signature. The method can have parameters or return simple types or objects that conform to the java bean specification.
For example, the `addUser()` method maps a user from the given LDAP realm to a set of roles.

```
public void addUser(String realm, LdapUserMapping config) {
    // the code
}
```

2. Add annotations to mark the method as an operation and define its parameters.

Annotations	Description
TeaOperation	Marks the method as an operation. The annotation defines the name and description of the operation.
KeyParam	Each operation is invoked in the context of the object, KeyParam. It is a key of the parent object. This attribute can be skipped for an operation on top-level types, as there can be only one instance of top level object per agent.
TeaParam	Each parameter is annotated with a parameter name, an optional description and a default value. This is required as Java does not preserve names of the parameters at runtime.
TeaRequires (optional)	Lists all the permissions required to invoke the operation in addition to the Read permission for the parent object. If the annotation is not present, anyone who has Read permission for the parent object will be able to invoke it.



To return a JSON object, the agent library supports Jackson databind, Jackson and JAXB annotations. The agent library does not support org.json module for Jackson. You can write plain java objects with annotation, or if you want to use generic types, you can use the Jackson databind API or use `Map<String, Object>`, which is a map of maps representing the JSON structure.

For example, the `addUser()` method is marked as an operation using the `TeaOperation` annotation.

```
@TeaRequires("TEA_ADMIN")
@TeaOperation(name = "add-user", description = "Create new user")
public void addUser(
    @KeyParam final String key,
    @TeaParam(name = "realm") final String realm,
    @TeaParam(name = "user") final Object config) {
    // code
}
```

The annotation is sufficient to expose this operation in the default UI. The operation bar on the object has a button for adding a user which opens a default form. The operation is available in the shell as well.

3. Add parameters that represent file artifacts. Use the `TeaParam` annotation on a method parameter with type `javax.activation.DataSource`. For example:

```
@TeaOperation(name = "upload", description = "Upload File")
public String uploadFile(
    @TeaParam(name = "file", description = "File to Upload") DataSource upload)
    throws IOException {

    InputStream inputStream = upload.getInputStream();
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    byte[] buffer = new byte[1024];
    int read;
    while ((read = inputStream.read(buffer)) != -1) {
        outputStream.write(buffer, 0, read);
    }
    inputStream.close();
    outputStream.close();
    File file = File.createTempFile(upload.getName(), null);
    FileOutputStream fileOutputStream = new FileOutputStream(file);
    outputStream.writeTo(fileOutputStream);
    fileOutputStream.close();
    return upload.getName();
}
```

4. Specify the permissions required to invoke the operation. The `TeaRequires` annotation lists all the permissions required to invoke the operation in addition to the `Read` permission for the parent object. If the annotation is not present, anyone with the `Read` permission for the parent object can invoke it. For example:

```
@TeaRequires("TEA_ADMIN")
@TeaOperation(name = "add-user", description = "Create new user")
public void addUser(
    @KeyParam final String key,
    @TeaParam(name = "realm") final String realm,
    @TeaParam(name = "user") final Object config) {
    // code
}
```

The Permission names (in this case, "TEA_ADMIN") are defined by the Agent, which is displayed in the UI for creating privileges and roles.



If there are two agents for the same object type, ensure that they have the same operation name and number. This is to ensure that when you invoke an operation, you can select the agent on which you want to execute the operation from the drop-down list on the Web-based GUI.

Asynchronous Operations Support

For defining an asynchronous operation, you should return an instance of `com.tibco.tea.agent.api.SettableFuture`.

`SettableFuture` is a custom `Future` object which the TIBCO Enterprise Administrator agent library uses for tracking asynchronous operations.

For example,

```
public SettableFuture<Response> startProcess {
    SettableFuture<Response> future = SettableFutureFactory.create();
    // pass future to the code which will handle the operation execution
    return future;
}
```

Where,

- `startProcess` is an asynchronous operation which has to return an object of type `Response`. To use the asynchronous support provided by the TIBCO Enterprise Administrator agent library, `startProcess` sets the return type as `SettableFuture<Response>`.
- `startProcess` will start by creating an instance of `SettableFuture` using the `SettableFutureFactory`.
- The `SettableFuture` instance is passed to the code which handles the operation execution. For example, the `SettableFuture` instance might be passed on to an instance of a class which implements `java.lang.Runnable` and that instance is passed to an instance of `java.util.concurrent.Executor`. Hence the starting of the process is handled by a thread pool.
- Finally, the instance of `SettableFuture` is returned to the TIBCO Enterprise Administrator agent library. The TIBCO Enterprise Administrator agent library tracks the asynchronous operation by following this instance.

Using `SettableFuture` instance to set the progress

You can also use the instance of `SettableFuture` for setting the progress, progress status, and returning the object. For example,

```
// inside the code handling operation execution
future.setProgressStatus("Connecting to host to start the process");
// ...
future.setProgress(25);
future.setProgressStatus("Spawning the process");
// ...
future.setProgress(50);
future.setProgressStatus("Waiting for the process to come up");
// ...
future.setProgress(100);
future.set(response);
```

Where,

- `SettableFuture.setProgressStatus()` is used to update the message which is displayed to the user to indicate the status of the asynchronous operation.
- `SettableFuture.setProgress()` is used to update the percentage of work that is completed so far. This number should be between 0 and 100.
- `SettableFuture.set()` is used for setting the response object which is returned to the TIBCO Enterprise Administrator server. This indicates the completion of the asynchronous operation.

Using `SettableFuture` instance for exception

You can also use the instance of `SettableFuture` in case of an exception.

```
// inside the code handling operation execution
future.setException(ex);
```

Where, `SettableFuture.setException()` should be used for setting the exception object that is returned to the TIBCO Enterprise Administrator server. This indicates the completion of the asynchronous operation.

Defining References for Object Types with Interfaces

A method on one of the registered object types can be exposed as a reference available to TIBCO Enterprise Administrator server and the user interface. In this manner, you can refer a TeaObject from another TeaObject.

Procedure

1. Define the method signature. The method cannot have any parameters and must return an array of `com.tibco.tea.agent.api.BaseTeaObject` objects.

For example, the `getNodes()` method returns an array of nodes, which implements the class `com.tibco.tea.agent.api.BaseTeaObject`.

```
public Node[] getNodes(){
    // The code...
}
```

2. Mark the method as a reference using the `@TeaReference` annotation. It provides the following information:

Parameters	Description
name	Name of the reference
referenceType (optional)	Identifies the type of an element when an <code>AgentObjectIdentifier</code> is used
objectType (optional)	Needed when more than one object types are defined on the referenced class

For example:

```
@TeaReference(name = "nodes", referenceType = "nodeType", objectType =
"nodeObjectType")
public 2Node[] getNodes(){
    // The code...
}
```

Defining References for Object Types with Annotations

A method on one of the registered objects type, created using annotations, can be exposed as a reference available to TIBCO Enterprise Administrator server and the user interface. In this manner, you can refer to a TeaObject from another TeaObject.

Procedure

1. Define a method, which has an object key as a parameter and returns an array of `com.tibco.tea.agent.types.AgentObjectIdentifier` objects.

Each reference is invoked in context of the object and the parameter key identifies the source object. This attribute can be skipped for references on top-level types, as there can be only one instance of a top level object per agent.

For example, the `getApplications()` method returns an array of `com.tibco.tea.agent.types.AgentObjectIdentifier` references.

```
public AgentObjectIdentifier[] getApplications(final String key){
    //The code...
}
```

2. Add the `TeaReference` annotation.

For example, the `getApplications()` method is marked as a reference using the `TeaReference` annotation.

```
@TeaReference(name = "applications", referenceType = "applicationType",
objectType = "applicationObjectType")
public AgentObjectIdentifier[] getApplications( final String key){
    //The code...
}
```

Code explanation:

- The `TeaReference` annotation marks the method as a reference. It gives you the following information:

Parameters	Description
name	Name of the reference
referenceType (optional)	Type of the element when an <code>AgentObjectIdentifier</code> is used
objectType (optional)	Needed when more than one object type is defined on the referenced class

- The `getApplications()` method returns an array of `AgentObjectIdentifier` references.
- Each reference is invoked in context of the object, `KeyParam`, which is a key of the source object. This attribute can be skipped for references on top-level types, as there can be only one instance of top level object per agent.

Object ID

The object ID provides information about the object and the associated agent.

All TIBCO Enterprise Administrator object IDs have the following structure:

```
<agentID>:<agentType>:<agentVersion>:<objectType>:<objectKey>
```

The agentId, agentType, agentVersion, objectType and objectKey tokens are URL encoded and the character colon ':' is allowed in those tokens.

Object ID tokens

agentID

Specifies the agent that owns a requested object or collection.

Ensure that the Agent IDs are reproducible using only the sort of information that external applications use to identify related objects. For example, an EMS server would base its Object id either on the JNDI name of the connection factory or on the connection URL (possibly with a well-known, hard-coded prefix such as "jndi:" or "url:"). Avoid using Agent IDs that contain random numbers, internal-use-only keys or other difficult-to-reproduce information. Agent developers must address this issue to support pivoting.

An effective agent IDs must not begin with "_". All strings beginning with "_" are reserved for TIBCO Enterprise Administrator.

agentType

Is the name of the Agent type.

Agent type names must not begin with "_". All strings beginning with "_" are reserved for TIBCO Enterprise Administrator.

agentTypeVersion

Is the version of the Agent type.

objectType

Is the name of the Object type.

Object type names must not begin with "_". All strings beginning with "_" are reserved for TIBCO Enterprise Administrator. The name "agent" is reserved for use by TIBCO Enterprise Administrator.

objectKey

An object key is specified to show details about a specific object instance.

The object key is an opaque string. The pair (agentID, objKey) must be unique among all objects that share the same pair (agent type, object type).

An effective object key should not begin with "_". All strings beginning with "_" are reserved for TIBCO Enterprise Administrator.

Solution

A *solution* defines a set of managed objects that can be managed by agents other than the one defining the solution.

A solution can contain objects defined by the agent and can have links to other objects. A Solution must be registered before an agent is started.

For example,

```
final TeaSolution solution = new TeaSolution("sampleSolution", "This is my sample
solution");
    // Add tomcat reference
    final TeaObjectHardLink hl = new TeaObjectHardLink() {

        @Override
        public String getName() {
            return "Tomcat";
        }

        @Override
        public String getDescription() {
            return "Link to tomcat";
        }

        @Override
        public String getObjectID() {
            return "Tomcat::server:t1";
        }
    };
    solution.addMembers(devNode, platformapp, hl);
    server.registerSolution(solution);
    server.start();
```

Roles

A *role* is a collection of privileges that are assigned to users or groups using class-level annotations.

Before defining roles on an agent, ensure that the agent is registered with the server. While defining roles, if there is a conflict in the names, the roles available on the server are used. You can change or delete roles imported to the server. When the last agent of a specific type is unregistered, associated roles are removed.

Roles can be assigned using class-level annotations. For example,

```
@TeaRoles({
    @TeaRole(name = "Tomcat Admin", desc = "Manage all tomcat servers",
        privileges = { @TeaPrivilege(permissions =
            { TeaPrivilege.FULL_CONTROL }) }),
    @TeaRole(name = "Tomcat User", desc = "Read only access to all tomcat
        servers", privileges = { @TeaPrivilege(permissions = {
            TeaPrivilege.READ, TomcatAgent.UPDATE_PERMISSION }) }) })
})

public class TomcatServer {

    @TeaRequires("Full Control")
    public void changePort(@KeyParam final String key,
        @TeaParam(name = "port", description = "New port number to use")
        @Customize(value = "label=Port")
        final int port) throws TeaIllegalArgumentException {
        // code
    }
}
```

As shown in the example, two roles are assigned: one for the Tomcat Administrator and one for a regular user.

TeaRole

TeaRole annotation is used to assign the default roles provided by the agent. The roles are available in the TIBCO Enterprise Administrator server only after registering an agent of a specific agent type for the first time. They are removed when the last agent of a specific type is unregistered. After creation, roles can be changed by the TIBCO Enterprise Administrator SDK Server administrator.

TeaRoles

If a specific role already exists on the server, it will be ignored. To assign multiple roles by the same class, use the grouping annotation, TeaRoles.

TeaPrivilege

Defines the privileges of a role. A *privilege* describes a set of permissions that are granted to objects that match the specified path pattern. A role can have one or more privileges associated with it.

The following attributes are available on the TeaPrivilege annotation :

pattern

Matches the paths (of objects) to which the privileges must be applied. The pattern expression can contain a * or **. * matches except the character / which is a path separator. The pattern ** matches everything. In the example, the Tomcat User role has Read permissions on all objects that are members of the object that contains tomcat in the path.

type

An optional attribute that indicates whether or not permissions should be granted. A type can take one of the two values : allow to grant permissions for a role or deny to deny permissions for a role. The default value for the type is allow.

permissions

A list of permissions that are applicable to this role. Some default permissions are available on the server. Agents can add more to this list. By default, the `Full Control` (full access to all objects and operations) and `Read` (read-only access) permissions are available on the server.

Permissions

TIBCO Enterprise Administrator implements permission checking based on the privileges and the roles defined on an object.

Key terms

User

Users are entities that need access to the system. Each user might need a different level of access to the system. Users can be assigned to multiple Roles. TIBCO Enterprise Administrator does not manage users by itself. Users from external systems are mapped into TIBCO Enterprise Administrator to allow access to the system.

Group

Groups are logical groupings of the users within an organization. A user can belong to multiple groups and a group can contain multiple users. Groups provide an easier way to control access to users. Instead of specifying the access permissions for each user, it is easier and practical to specify access permissions to the groups to which they belong to. Groups can contain sub-groups.

Realm

A security realm comprises mechanisms for protecting TIBCO Enterprise Administrator resources. It contains users, groups, and their security credentials. The realm provides information about users and the groups they belong to. TIBCO Enterprise Administrator supports two kinds of realms : File and LDAP. In a File realm, the user and group information is stored in a file. In an LDAP realm, the user or group information exists on an LDAP server and is accessed from the server.

Permission

A string on the basis of which access control is enforced. It is upto the agent to decide the granularity of the permissions that it provides. For example, a permission could be as fine-grained as 'UpdateConfig' which is applicable to only one operation, or it could be as coarse-grained as 'Full Control' which applies to the entire system.

Privilege

Privilege is a collection of permissions that are applicable to an object or a collection of objects.

Role

Role is a mechanism to grant or revoke access to users. A Role is a collection of privileges and are assigned to users and groups. All the privileges in a role get associated to the user or group to which it is assigned.

Custom Permission

You can assign custom permissions by using the TeaPermission and TeaPermissions annotation.

For example, Lifecycle and Update permission are grouped using the TeaPermissions annotation.

```
@TeaObjectType(name = TomcatAgentUtil.TOMCAT, concept = TeaConcept.TOP_LEVEL,
    description = "Tomcat TIBCO Enterprise Administrator SDK Agent")
@TeaPermissions({
    @TeaPermission(name = TomcatAgent.LIFECYCLE_PERMISSION,
        desc = "Permission to create/start/stop server, webapp"),
    @TeaPermission(name = TomcatAgent.UPDATE_PERMISSION,
        desc = "Permission to update configurations of server, webapp") })

public class TomcatAgent {
    // code
}
```

An agent can define the permissions needed to execute each of the operations that it provides. If a method does not have any TeaRequires annotation on it, then that method can be executed by anyone.

Effective Permissions

The collection of privileges that are applicable to a user are obtained as follows:

- Gather the privileges from all the roles assigned to this user directly.
- Gather the privileges from the roles assigned to all the groups to which the user belongs.

While checking for permissions, all the privileges that match the path of an object are found and evaluated for one role at a time.

Some privileges could be conflicting because some of these privileges are of the `allow` type and some of them are of the `deny` type. The following algorithm is used for permission checking while evaluating the privileges within a role:

- If at least one matching role has the `allow` privilege, containing all the permissions needed by the object, access is granted.
- If a `deny` privilege matches the given object and the privilege has at least one permission needed by the operation, the role is not considered.
- If one matching `allow` privilege has the `full control` permission, access is granted.

User Interface Customization

The UI can be customized using an external file. The file can be uploaded through the `customize` operation on the agent type.

The file format of the customization file is JSON. If the customization file is provided as a part of the custom static resource, the file must be named `customization.json` and placed in the top-level folder. The file should contain a single object. The object members are individual customization rules, where the member name is the selected attribute and the member value is the customization value.

Syntax for the customization rules:

```
<Object Type Name>[#<Operation Name>[#<Method Type>[.<Parameter Name>]]]
```

The following is an example for customizing the references in the Tomcat sample provided with the product:

```
"server@members": {
  "title": "Tomcat",
  "columns": [
    { "label": "name", "expr": "name", "entityLink": true },
    { "label": "url", "expr": "config.path" },
    { "label": "status", "expr": "status.state" }
  ]
}
```

code explanation:

- `server@members`: is the reference for the object type `server`
- `title`: "Tomcat": Is the custom title which can also be defined using the "label" flag
- `columns`: Indicates the number of columns. In this case, it is 3.
 - `label`: Is the title of the column.
 - `expr`: Is the content in the column.
 - `entityLink`: Is a toggle that can be either `true` or `false`. Indicates that a cell can be displayed as a link to an entity page.

The following additional properties can be set:

- `collapsed`: This property can be set on a reference. If set, the panel containing the reference will be collapsed, on display.
- `referenceOrder`: Is the order in which a reference is displayed. For example, `"referenceOrder": ["foo", "bar"]` indicates that `"foo"` is displayed before `"bar"`.
- `confirm`: This property can be set on an operation. If set to `false`, the confirmation dialog is skipped.

Configuring TIBCO Enterprise Administrator Agent for Auto-Registration

The TIBCO Enterprise Administrator agent library can be configured to auto-register itself with the TIBCO Enterprise Administrator server. When TIBCO Enterprise Administrator Agent comes up, the agent library connects to the server and registers itself. Ensure that the agent library is setup with the correct connection details for the server.

Procedure

1. Configure the authentication for connecting with the server using `java.net.Authenticator`.

```
Authenticator.setDefault(new Authenticator() {
    /* (non-Javadoc)
     * @see java.net.Authenticator#getPasswordAuthentication()
     */
    @Override
    protected PasswordAuthentication
getPasswordAuthentication() {
    return new PasswordAuthentication("admin",
"admin".toCharArray());
    }
});
server.registerInstance(new HelloWorldAgent());
```

2. `com.tibco.tea.agent.server.TeaAgentServer.registerAgentAutoRegisterListener()` should be used for specifying that the agent should register with the server and the *serverUrl* where the server is accessible.

```
server.registerAgentAutoRegisterListener("http://localhost:8777/tea");
server.start();
```

Example: Configuring TeaAgentServer for Auto-registration

This example shows how `TeaAgentServer` can be configured to auto-register itself. You can use the same procedure to configure `TeaAgentServlet` to auto-register with the server.

```
TeaAgentServer server = new TeaAgentServer("HelloWorldAgent", "1.0", "Hello
World Agent", 1234, "/helloworldagent", true);
Authenticator.setDefault(new Authenticator() {
    @Override
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication("admin", "admin".toCharArray());
    }
});
server.registerInstance(new HelloWorldAgent());
server.registerAgentAutoRegisterListener("http://localhost:8777/tea");
server.start();
```

Unregistering an Agent: The agent is unregistered by using the `TeaAgentServer.unregisterAgent()` method. Similar to registration, authentication is configured using the `java.net.Authenticator` method.