Template version: proc-RFC_template-1_00_001117.dot

RFC 11 ServiceTracker

Members Only, Final
cpeg-rfc_11_ServiceTracker-1.00

16 Pages

Abstract

While the framework in the OSGi core platform provides a powerful framework for the bundle programmer, many times the flexibility of the framework makes it difficult for the programmer to correctly use the service registry. The ServiceTracker class deals with the subtleties of using the services: tracking service events, getting and ungetting services, race conditions, etc. ServiceTracker is designed to fulfil the basic needs of the bundle programmer when using services.

0 Document Information

0.1 Table of Contents

All Page Within This Box

0.2 Status

This document specifies the ServiceTracker class and related ServiceTrackerCustomizer interface for the core platform of the Open Services Gateway Initiative, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

0.3 Acknowledgement

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

```
Source code is shown in this typeface.
```

0.5 Revision History

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

Revision	Date	Comments
Review Draft 1	Jan 23 2001	Updated Javadoc and converted RFC to Word format. Includes changes suggested at London CPEG meeting. BJ Hargrave, IBM, Core Platform Expert Group, OSGi. hargrave@us.ibm.com
Review Draft 2	Jan 26 2001	Added remove method to ServiceTracker. BJ Hargrave, IBM, Core Platform Expert Group, OSGi. hargrave@us.ibm.com
Review Draft 3	Feb 08 2001	Addressed comments from Jan Luehe, Sun. BJ Hargrave, IBM, Core Platform Expert Group, OSGi. hargrave@us.ibm.com
Final	Mar 07 2001	Changed package name to org.osgi.util.tracker. Added statement that bundles must import the org.osgi.util.tracker package to use the tracker. BJ Hargrave, IBM, Core Platform Expert Group, OSGi. hargrave@us.ibm.com

All Page Within This Box

1 Introduction

The ServiceTracker class provides the bundle with a simple way to perform the common operations necessary for using OSGi services. The tracker allows Bundle programmers to specify the services they are interested in tracking using a class name, a Filter or a ServiceReference. The ServiceTracker can be customized by subclassing ServiceTracker or implementing the ServiceTrackerCustomizer interface.

The basic tasks of the ServiceTracker are:

- To listen to ServiceEvents so that services of interest to the bundle programmer are properly tracked.
- To allow the bundle programmer to customize the tracking process by allowing her to programmatically select which services are to be tracked as well as take action when a service is added or removed.

In summary, the ServiceTracker provides an easy way to accomplish the tasks that most users of a service need.

2 Motivation and Rationale

The service registry in the OSGi framework is a very powerful tool that has proved difficult for the average bundle programmer to use. Even experienced OSGi programmers may miss race conditions or boundary conditions that will lead to random errors. To overcome these problems, programmers find themselves writing the same code over and over. The intent of this RFC is to provide programmers a tool for accessing services in a simple way that is flexible enough to cover most needs of a service user.

3 Technical Discussion

The OSGi ServiceTracker Package. Specification Version 1.1.

Bundles wishing to use this package must list the package in the `Import-Package` header of the bundle's manifest. For example

All Page Within This Box

```
Import-Package: org.osgi.util.tracker; specification-version=1.1
```

3.1 ServiceTracker

The ServiceTracker class watches for a set of services that match given search criteria and watches the ServiceEvents corresponding to those services.

The behavior of the ServiceTracker may be customized by specifying a ServiceTrackerCustomizer object when creating a ServiceTracker. The ServiceTrackerCustomizer methods, addingService, modifiedService and removedService, are called whenever a service is being added to the ServiceTracker or when a tracked service is modified or removed from the ServiceTracker.

The ServiceTracker may alternatively be customized by subclassing and overriding the addingService, modifiedService and removedService methods. These methods on ServiceTracker are called when the ServiceTracker is created using the default constructor.

3.1.1 Starting a ServiceTracker

A ServiceTracker can be created in one of three ways. There are three constructors to create a ServiceTracker each providing a different search criteria.

- The first is to specify a class name as the search criteria. The ServiceTracker will then track all services that are registered under the specified class name.
- Second, services are registered with a set of properties. A Filter object can be used to specify the search criteria. The ServiceTracker will then track all services that have properties that match the specified filter.
- Finally, a ServiceReference can also be specified to track a specific service. The ServiceTracker will then track only the service that corresponds to the specified ServiceReference. In this case there will never be more than one service tracked, since a ServiceReference refers to a single service.

Once a ServiceTracker is constructed, it will not begin tracking services until the open method is called. Note that each of the constructors takes a BundleContext as a parameter. This BundleContext will be used by the ServiceTracker to track, get, and unget the services.

3.1.2 Customizing the ServiceTracker

Once the ServiceTracker has been opened, it will automatically track the services matching the specified search criteria. When a service is being added to the tracker or when a tracked service is modified or removed from the ServiceTracker, the ServiceTracker will call the addingService, modifiedService, or removedService methods, respectively, on the ServiceTrackerCustomizer object, if specified when the ServiceTracker was created, or on the ServiceTracker itself. The ServiceTracker class implements the ServiceTrackerCustomizer interface to provide a default ServiceTrackerCustomizer behavior.

So the behavior of the ServiceTracker can be customized by either specifying a ServiceTrackerCustomizer object implementing the desired behavior when the ServiceTracker is constructed or by subclassing ServiceTracker and overriding the ServiceTrackerCustomizer methods.

When a service matching the search criteria is located and begins to be tracked, the ServiceTracker will call the ServiceTrackerCustomizer.addingService method. The default behavior of the addingService

method of the ServiceTracker class is to get and return the service object (`BundleContext.getService`) for the service being added.

A bundle programmer may wish to customize the action when a service is tracked. An example of this may be a Servlet that will be registered with each HttpService that is tracked. This could be done by specifying a ServiceTrackerCustomizer object and implementing addingService and removedService as follows:

```
public Object addingService(ServiceReference reference)
{
        Object obj = context.getService(reference);
        HttpService svc = (HttpService)obj;
        ...
        // Register the Servlet using svc
        ...
        return svc;
}

public void removedService(ServiceReference reference, Object obj)
{
        HttpService svc = (HttpService)obj;
        ...
        // Unregister the Servlet using svc
        ...
        context.ungetService(reference);
}
```

An alternate way to perform this customization would be to subclass ServiceTracker and override the same methods.

Another reason for customizing a ServiceTracker is to programmatically select which services are tracked. A filter may not sufficiently specify the services that the programmer is interested in. By implementing the addingService method, the programmer can use additional runtime information to determine if the service should be tracked. If `null` is returned by addingService, the service will not be tracked.

Finally, the bundle programmer can return any object from the addingService method, it doesn't have to be the object that resulted from the `BundleContext.getService` call. In some cases the bundle program may instantiate another class using the service retrieved from the framework and return that as the object to be tracked. When the removedService method is called, the object that is passed along with the ServiceReference is the object that was returned from the earlier addingService method.

3.1.3 Using the ServiceTracker

Once the bundle programmer has started the ServiceTracker there are a number of methods available to access the services that are being tracked. The simplest is the getService method. This method returns one of the services being tracked or `null` if there are no active services being tracked. A more complex form of this method is getServices which returns an array of the all the tracked services. The number of tracked services is returned by the size method.

All Page Within This Box

Along with the services, often it is useful to get the ServiceReferences for the tracked services. The getServiceReferences method returns a list of the ServiceReferences for the tracked services. The service object for a specific tracked service may be returned by calling the ServiceTracker's getService(ServiceReference) method.

The waitForService method allows the caller to wait until at least one instance of a service is tracked or until the timeout expires. If the timeout is zero, the caller will wait until at least one instance of a service is tracked. It is strongly recommended that waitForService is not used during the BundleActivator methods. BundleActivator methods are expected to complete in a short period of time.

The remove method may be used to remove a specific service from being tracked by the ServiceTracker. This results in removedService being called for that service.

The close method will remove all services being tracked by the ServiceTracker. This results in removedService being called for all of the tracked services.

3.2 Class ServiceTracker

```
java.lang.Object
   |
   +-org.osgi.util.tracker.ServiceTracker
```

All Implemented Interfaces:

ServiceTrackerCustomizer

---

public class **ServiceTracker**
extends java.lang.Object
implements ServiceTrackerCustomizer

The ServiceTracker simplifies using services from the Framework's service registry.

A ServiceTracker is constructed with search criteria and a ServiceTrackerCustomizer object. The ServiceTracker can use the ServiceTrackerCustomizer to customize the service objects to be tracked. The ServiceTracker can then be opened to begin tracking all services in the framework's service registry that match the specified search criteria. The ServiceTracker correctly handles all of the details of listening to ServiceEvents and getting and ungetting services.

The getServiceReferences method can be called to get references to the services being tracked. The getService and getServices methods can be called to get the service objects for the tracked service.

Since:

1.1

Field Summary	
protected BundleContext	**context** Bundle context this ServiceTracker is tracking against.
protected Filter	**filter** Filter specifying search criteria for the services to track.

All Page Within This Box

Constructor Summary
ServiceTracker(BundleContext context, Filter filter, ServiceTrackerCustomizer customizer) Create a ServiceTracker on the specified Filter.
ServiceTracker(BundleContext context, ServiceReference reference, ServiceTrackerCustomizer customizer) Create a ServiceTracker on the specified ServiceReference.
ServiceTracker(BundleContext context, java.lang.String clazz, ServiceTrackerCustomizer customizer) Create a ServiceTracker on the specified class name.

Method Summary	
java.lang.Object	**addingService**(ServiceReference reference) Default implementation of the ServiceTrackerCustomizer.addingService method.
void	**close**() Close this ServiceTracker.
protected void	**finalize**() Properly close this ServiceTracker when finalized.
java.lang.Object	**getService**() Returns a service object for one of the services being tracked by this ServiceTracker.
java.lang.Object	**getService**(ServiceReference reference) Returns the service object for the specified ServiceReference if the referenced service is being tracked by this ServiceTracker.
ServiceReference[]	**getServiceReferences**() Return an array of ServiceReferences for all services being tracked by this ServiceTracker.
java.lang.Object[]	**getServices**() Return an array of service objects for all services being tracked by this ServiceTracker.
void	**modifiedService**(ServiceReference reference, java.lang.Object service) Default implementation of the ServiceTrackerCustomizer.modifiedService method.
void	**open**() Open this ServiceTracker and begin tracking services.
void	**remove**(ServiceReference reference) Remove a service from this ServiceTracker.
void	**removedService**(ServiceReference reference, java.lang.Object object) Default implementation of the ServiceTrackerCustomizer.removedService method.
int	**size**() Return the number of services being tracked by this ServiceReference.

All Page Within This Box

java.lang.Object	**waitForService**(long timeout) Wait for at least one service to be tracked by this ServiceTracker.

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

3.2.1 context

```
protected final BundleContext context
```

Bundle context this ServiceTracker is tracking against.

3.2.2 filter

```
protected final Filter filter
```

Filter specifying search criteria for the services to track.

Constructor Detail

3.2.3 ServiceTracker

```
public ServiceTracker(BundleContext context,
                      ServiceReference reference,
                      ServiceTrackerCustomizer customizer)
```

Create a ServiceTracker on the specified ServiceReference.

The service referenced by the specified ServiceReference will be tracked by this ServiceTracker.

Parameters:

`context` - Bundle context against which the tracking is done.

`reference` - ServiceReference to the service to be tracked.

`customizer` - The customizer object to call when services are added, modified, or removed in this ServiceTracker. If customizer is null, then the ServiceTracker itself will be used as the ServiceTrackerCustomizer object and the ServiceTracker will call the ServiceTrackerCustomizer methods on itself.

3.2.4 ServiceTracker

```
public ServiceTracker(BundleContext context,
                      java.lang.String clazz,
                      ServiceTrackerCustomizer customizer)
```

Create a ServiceTracker on the specified class name.

Services registered under the specified class name will be tracked by this ServiceTracker.

Parameters:

`context` - Bundle context against which the tracking is done.

`clazz` - Class name of the services to be tracked.

`customizer` - The customizer object to call when services are added, modified, or removed in this ServiceTracker. If customizer is null, then the ServiceTracker itself will be used as the ServiceTrackerCustomizer object and the ServiceTracker will call the ServiceTrackerCustomizer methods on itself.

3.2.5 ServiceTracker

```
public ServiceTracker(BundleContext context,
                      Filter filter,
                      ServiceTrackerCustomizer customizer)
```

Create a ServiceTracker on the specified Filter.

Services which match the specified Filter will be tracked by this ServiceTracker.

Parameters:
`context` - Bundle context against which the tracking is done.
`filter` - Filter to select the services to be tracked.
`customizer` - The customizer object to call when services are added, modified, or removed in this ServiceTracker. If customizer is null, then the ServiceTracker itself will be used as the ServiceTrackerCustomizer object and the ServiceTracker will call the ServiceTrackerCustomizer methods on itself.

Method Detail

3.2.6 open

```
public void open()
```

Open this ServiceTracker and begin tracking services.

Services which match the search criteria specified when this ServiceTracker was created are now tracked by this ServiceTracker.

Throws:
`java.lang.IllegalStateException` - if the BundleContext the ServiceTracker was created with is no longer valid.

3.2.7 close

```
public void close()
```

Close this ServiceTracker.

This method should be called when this ServiceTracker should end the tracking of services.

3.2.8 finalize

```
protected void finalize()
                throws java.lang.Throwable
```

Properly close this ServiceTracker when finalized. This method calls the close method to close this ServiceTracker if it has not already been closed.
Overrides:
`finalize` in class `java.lang.Object`

All Page Within This Box

3.2.9 addingService

```
public java.lang.Object addingService(ServiceReference reference)
```

Default implementation of the ServiceTrackerCustomizer.addingService method.

This method is only called when this ServiceTracker has been constructed with a `null` ServiceTrackerCustomizer parameter. The default implementation returns the result of calling getService, on the BundleContext with which this ServiceTracker was created, passing the specified ServiceReference.

This method can be overridden to customize the service object to be tracked for the service being added to this ServiceTracker.

Specified by:
addingService in interface ServiceTrackerCustomizer
Parameters:
`reference` - Reference to service being added to this ServiceTracker.
Returns:
The service object to be tracked for the service added to this ServiceTracker.

3.2.10 modifiedService

```
public void modifiedService(ServiceReference reference,
                            java.lang.Object service)
```

Default implementation of the ServiceTrackerCustomizer.modifiedService method.

This method is only called when this ServiceTracker has been constructed with a `null` ServiceTrackerCustomizer parameter. The default implementation does nothing.

Specified by:
modifiedService in interface ServiceTrackerCustomizer
Parameters:
`reference` - Reference to modified service.
`service` - The service object for the modified service.

3.2.11 removedService

```
public void removedService(ServiceReference reference,
                           java.lang.Object object)
```

Default implementation of the ServiceTrackerCustomizer.removedService method.

This method is only called when this ServiceTracker has been constructed with a `null` ServiceTrackerCustomizer parameter. The default implementation calls ungetService, on the BundleContext with which this ServiceTracker was created, passing the specified ServiceReference.

Specified by:
removedService in interface ServiceTrackerCustomizer
Parameters:
`reference` - Reference to removed service.
`service` - The service object for the removed service.

All Page Within This Box

3.2.12 waitForService

```
public java.lang.Object waitForService(long timeout)
                                throws java.lang.InterruptedException
```

Wait for at least one service to be tracked by this ServiceTracker.

It is strongly recommended that waitForService is not used during the BundleActivator methods. BundleActivator methods are expected to complete in a short period of time.

Parameters:
`timeout` - time interval in milliseconds to wait. If zero, the method will wait indefinately.
Returns:
Returns the result of getService().

3.2.13 getServiceReferences

```
public ServiceReference[] getServiceReferences()
```

Return an array of ServiceReferences for all services being tracked by this ServiceTracker.
Returns:
Array of ServiceReferences or `null` if no service are being tracked.

3.2.14 getServices

```
public java.lang.Object[] getServices()
```

Return an array of service objects for all services being tracked by this ServiceTracker.
Returns:
Array of service objects or `null` if no service are being tracked.

3.2.15 getService

```
public java.lang.Object getService(ServiceReference reference)
```

Returns the service object for the specified ServiceReference if the referenced service is being tracked by this ServiceTracker.
Parameters:
`reference` - Reference to the desired service.
Returns:
Service object or `null` if the service referenced by the specified ServiceReference is not being tracked.

3.2.16 getService

```
public java.lang.Object getService()
```

Returns a service object for one of the services being tracked by this ServiceTracker.
Returns:
Service object or `null` if no service is being tracked.

3.2.17 remove

```
public void remove(ServiceReference reference)
```

All Page Within This Box

Remove a service from this ServiceTracker. The specified service will be removed from this ServiceTracker. If the specified service was being tracked then the ServiceTrackerCustomizer.removedService method will be called for that service.

Parameters:

`reference` - Reference to the service to be removed.

3.2.18 size

```
public int size()
```

Return the number of services being tracked by this ServiceReference.

Returns:

Number of services being tracked.

3.3 Interface ServiceTrackerCustomizer

All Known Implementing Classes:

ServiceTracker

---

public interface **ServiceTrackerCustomizer**

The ServiceTrackerCustomizer interface allows the ServiceTracker client to customize the service objects that are tracked by the ServiceTracker. The ServiceTrackerCustomizer is called when service is being added to the ServiceTracker. The ServiceTrackerCustomizer can then return an object for the tracked service. The ServiceTrackerCustomizer is also called when a tracked service is modified or has been removed from the ServiceTracker.

Since:

1.1

Method Summary

java.lang.Object	**addingService**(ServiceReference reference) A service is being added to the ServiceTracker.
void	**modifiedService**(ServiceReference reference, java.lang.Object service) A service tracked by the ServiceTracker has been modified.
void	**removedService**(ServiceReference reference, java.lang.Object service) A service tracked by the ServiceTracker has been removed.

Method Detail

3.3.1 addingService

```
public java.lang.Object addingService(ServiceReference reference)
```

A service is being added to the ServiceTracker.

This method is called before a service which matched the search parameters of the ServiceTracker is added to the ServiceTracker. This method should return the service object to

All Page Within This Box

be tracked for this ServiceReference. The returned service object is stored in the ServiceTracker and is available from the getService and getServices methods.

Parameters:
`reference` - Reference to service being added to the ServiceTracker.
Returns:
The service object to be tracked for the ServiceReference or `null` if the ServiceReference should not be tracked.

3.3.2 modifiedService

```
public void modifiedService(ServiceReference reference,
                            java.lang.Object service)
```

A service tracked by the ServiceTracker has been modified.

This method is called when a service being tracked by the ServiceTracker has had it properties modified.

Parameters:
`reference` - Reference to service that has been modified.
`service` - The service object for the modified service.

3.3.3 removedService

```
public void removedService(ServiceReference reference,
                           java.lang.Object service)
```

A service tracked by the ServiceTracker has been removed.

This method is called after a service is no longer being tracked by the ServiceTracker.

Parameters:
`reference` - Reference to service that has been removed.
`service` - The service object for the removed service.

4 Security Considerations

ServiceTracker runs in the security context of the bundle using it. It doesn't provide or remove any of the security checks that are already in place for bundles. It should be noted that since the ServiceTracker contains references to services care should be taken to only pass a ServiceTracker to classes that are trusted to use those services.

5 Document Support

5.1 References

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

5.2 Author's Address

Name	Benjamin Reed
Company	IBM Research
Address	650 Harry Road San Jose, CA USA
Voice	+1 408 927 1811
e-mail	breed@almaden.ibm.com

Name	Peter Kriens
Company	Ericsson
Address	Finnasandsvägen 22 Onsala 43933 Sweden
Voice	+46 300 39800
e-mail	Peter.Kriens@aQute.se

Name	BJ Hargrave
Company	IBM Pervasive Computing
Address	11400 Burnet Road Austin, TX 78758 USA
Voice	+1 512 838 9938
e-mail	hargrave@us.ibm.com

5.3 Acronyms and Abbreviations

All Page Within This Box

5.4 End of Document

All Page Within This Box