Shared Data Toolkit for Java Technology Implementation Guide

This is a toolkit defined to support highly interactive, collaborative applications written in the Java programming language. This guide describes how to add a new implementation beneath this API framework.

Version 2.3 November 14, 2017

Please send technical comments on this implementation guide to:

jsdt.interest@gmail.com

Rich Burridge, JSDT Author

Contents

1 Introduction

1.1 Overview

The JSDT API framework has been designed to be easily extended with new implementations. Or to put it another way, you can add alternate transport protocols beneath this toolkit, in a simple manner.

1.2 Alternate Socket Factories

The JSDT `socket` implementation has been designed to allow alternate socket factory classes to be specified. This implementation is unicast, and requires that each of those sockets can maintain a permanent connection to the server socket.

If you can wrap your transport protocol in a `java.net.socket` implementation, then this is the suggested approach. Your socket factory needs to create a default constructor and implement four methods which are specified in the `com.sun.media.jsdt.impl.JSDTSocketFactory` interface:

```
/**
 * createSocket returns a socket connected to a ServerSocket on
 * the named host, at the given port.  This socket is configured using
 * the socket options established for this factory.
 *
 * @param host the server host
 * @param port the server port
 *
 * @exception IOException if the connection can't be established
 * @exception UnknownHostException if the host is not known
 */

    Socket
    createSocket(String address, int port)
                throws IOException, UnknownHostException;

/**
 * createServerSocket returns a server socket which uses all network
 * interfaces on the host, and is bound to the specified port.
 *
 * @param port the port to listen to
 * @exception IOException for networking errors
 */
```

```
    ServerSocket
    createServerSocket(int port) throws IOException;


/**
 * createDatagramSocket returns a socket for sending and
 * receiving datagram packets, bound to any available port on
 * the local host machine. This socket is configured using the socket
 * options established for this factory.
 *
 * @exception SocketException if the socket could not be opened,
 * or the socket could not bind to the specified local port.
 *
 * @see java.net.DatagramSocket
 *
 * @return a DatagramSocket on any available local port.
 */

    DatagramSocket
    createDatagramSocket() throws SocketException;


/**
 * createDatagramSocket returns a socket for sending and
 * receiving datagram packets, at the given local port. This
 * socket is configured using the socket options established for
 * this factory.
 *
 * @param port the local port to use
 *
 * @exception SocketException if the socket could not be opened,
 * or the socket could not bind to the specified local port.
 *
 * @see java.net.DatagramSocket
 *
 * @return a DatagramSocket on the given local port.
 */

    DatagramSocket
    createDatagramSocket(int port) throws SocketException;
```

You then specify this alternate factory by setting the `com.sun.media.jsdt.impl.JSDTObject.socketFactoryClass` variable. For example:

```
com.sun.media.jsdt.impl.JSDTObjectsocketFactoryClass =
                                "myAlternateSocketFactory";
```

1.3 The URL String

The JSDT Session and Client URL's define which <*type*> implementation package will be dynamically loaded and used to run a JSDT collaboration. These are of the form:

```
Session:    jsdt://<host>:<port>/<type>/Session/<sessionName>
Client:     jsdt//<host>:<port>/<type>/Client/<clientName>
```

Each implementation is contained within it's own package. For the two existing implementations, these are:

sockets based com.sun.media.jsdt.socket

HTTP based com.sun.media.jsdt.http

A new <*type*> implementation type would have a package name of:

<*type*> based com.sun.media.jsdt.<*type*>

1.4 Hooks into the new Implementation Package

Instances of various classes in this new <*type*> package will be dynamically created as and when needed. These in turn will then be used to call various methods within those classes.

2 Using the template Package

2.1 Overview

Included with the implementers kit is a *com.sun.media.jsdt.template* package which can be used as a template for creating a new implementation. There are a set of steps that need to be followed to generate the standard package structure for a new implementation. These are described in the next section.

2.2 Creating the Standard Package Structure

For the purpose of these instructions, let's assume we are going to create a framework for a new *corba* implementation.

- Copy all the files in the template directory into a new *corba* directory.
- Three of the files need to be renamed to include the name of the implementation in the file name.
  - templateClient.java should be renamed to corbaClient.java
  - templateDebugFlags.java should be renamed to corbaDebugFlags.java
  - templateSession.java should be renamed to corbaSession.java

  The package name in all the files needs to changed from *com.sun.media.jsdt.template* to *com.sun.media.jsdt.corba*
- In corbaDebugFlags.java, the following changes should be made:
  - The interface line should be adjusted to read:

    ```
    interface corbaDebugFlags {
    ```

  - The last two lines should be changed to read:

    ```
    static final boolean corbaClient_Debug     = true;
    static final boolean corbaSession_Debug    = true;
    ```

- In corbaClient.java, the following changes need to be made:
  - The class lines should be adjusted to read:

    ```
    public class
    corbaClient extends ClientImpl
          implements ClientListener, corbaDebugFlags {
    ```

    All occurrences of *templateClient_Debug* should be replaced with *corbaClient_Debug*

    All occurrences of *templateClient* should be replaced with *corbaClient*. These will be the name of the constructor and the file name contained within various System.err.println debug messages.

- In corbaSession.java, the following changes need to be made:
  - The class lines should be adjusted to read:

    ```
    public class
    corbaSession extends SessionImpl implements corbaDebugFlags {
    ```

  - *All occurrences of templateSession_Debug should be replaced with corbaSession_Debug*
  - *All occurrences of templateSession should be replaced with corbaSession. These will be the name of the constructor and the file name contained within various System.err.println debug messages.*

3 Methods to Implement

3.1 Overview

As you will see, there are several methods that need to be completed for each new JSDT implementation. This chapter lists what those methods are. Note that some or all of these classes may also require you to write one or more constructors.

The JavaDoc for the *com.sun.media.jsdt.template* and *com.sun.media.jsdt.impl* packages should hopefully contain enough information on what actions each method should take. Additional documentation can be found in the JavaDoc for the *com.sun.media.jsdt* and *com.sun.media.jsdt.event* packages and the JSDT User Guide.

3.2 Standard Implementation Classes

The *com.sun.media.jsdt.impl* package comes with various classes that implement some of the interfaces in the *com.sun.media.jsdt* package. JavaDoc is provided with the JSDT Implementers kit which describes these classes. These classes are hopefully generic enough, such that instances of these classes should be useful to all JSDT implementations.

3.3 Client/Server Versus Fully-Replicated

Your implementation can be a Client/Server model, or it could be a fully-replicated distributed model. The supporting classes in the *com.sun.media.jsdt*, and *com.sun.media.jsdt.impl* packages provide hooks for creating server-side classes, which can be created then ignored for fully replicated implementations.

These hooks are in:

- ByteArrayServer.java
- ChannelServer.java
- ManageableServer.java
- SessionServer.java
- TokenServer.java

Each of these files defines two methods:

```
public void
initServer(String name, SessionImpl session, Object object);

public Object
getServer();
```

3.4 ByteArrayProxy

The ByteArrayProxy class contains the following methods that need to be completed for your implementation:

```
public void
initProxy(String name, SessionImpl session, Object object);

public Object
getProxy();

public void
setValue(Client client, byte[] value, int offset, int length)
        throws ConnectionException, InvalidClientException,
               NoSuchByteArrayException, NoSuchClientException,
               NoSuchSessionException, PermissionDeniedException,
               TimedOutException;
```

3.5 ChannelProxy

The ChannelProxy class contains the following methods that need to be completed for your implementation:

```
public void
initProxy(String name, SessionImpl session, Object object);

public Object
getProxy();

public void
addConsumer(Client client, ChannelConsumer consumer)
       throws ConnectionException, InvalidClientException,
              NoSuchChannelException, NoSuchClientException,
              NoSuchConsumerException, NoSuchSessionException,
              PermissionDeniedException, TimedOutException;

public void
removeConsumer(Client client, ChannelConsumer consumer)
       throws ConnectionException, InvalidClientException,
              NoSuchChannelException, NoSuchClientException,
              NoSuchConsumerException, NoSuchSessionException,
```

```
                PermissionDeniedException, TimedOutException;


public String[]
listConsumerNames()
        throws ConnectionException, NoSuchChannelException,
                NoSuchSessionException, TimedOutException;

public void
join(Client client, boolean authenticate, int mode)
        throws ConnectionException, InvalidClientException,
                NoSuchChannelException, NoSuchClientException,
                PermissionDeniedException, NoSuchSessionException,
                NameInUseException, TimedOutException;

public void
leave(Client client)
        throws ConnectionException, InvalidClientException,
                NoSuchChannelException, NoSuchClientException,
                NoSuchSessionException, TimedOutException;

public Data
receive(Client client, long timeout, boolean ignoreTimeout)
        throws ConnectionException, InvalidClientException,
                NoSuchClientException, NoSuchSessionException,
                PermissionDeniedException, TimedOutException;

public boolean
dataAvailable(Client client)
        throws ConnectionException, InvalidClientException,
                NoSuchClientException, NoSuchSessionException,
                PermissionDeniedException, TimedOutException;

public void
send(Client sendingClient, char recipient,
     String receivingClientName, Data data, boolean uniform)
        throws ConnectionException, InvalidClientException,
                NoSuchChannelException, NoSuchClientException,
                NoSuchConsumerException, NoSuchSessionException,
                PermissionDeniedException, TimedOutException;
```

3.6 ManageableProxy

The ManageableProxy class contains the following methods that need to be completed for your implementation:

```
public void
initProxy(String name, SessionImpl session, Object object);

public Object
getProxy();
```

```
public Session
getSession();

public void
addListener(EventListener listener, char listenerType)
       throws ConnectionException,
              NoSuchByteArrayException, NoSuchChannelException,
              NoSuchSessionException, NoSuchTokenException,
              TimedOutException;

public void
removeListener(EventListener listener, char listenerType)
       throws ConnectionException,
              NoSuchByteArrayException, NoSuchChannelException,
              NoSuchListenerException, NoSuchSessionException,
              NoSuchTokenException, TimedOutException;

public void
changeListenerMask(EventListener listener,
                   int eventMask, boolean disable)
       throws NoSuchSessionException, NoSuchChannelException,
              NoSuchByteArrayException, NoSuchTokenException,
              NoSuchListenerException;

public void
changeManagerMask(JSDTManager manager, int eventMask,
                  boolean disable, char objectType)
       throws ConnectionException, NoSuchSessionException,
              NoSuchChannelException, NoSuchByteArrayException,
              NoSuchTokenException, NoSuchListenerException,
              TimedOutException;

public void
attachManager(JSDTManager manager, char managerType,
              Manageable manageable)
       throws ConnectionException, ManagerExistsException,
              NoSuchHostException, NoSuchSessionException,
              TimedOutException;

public void
expel(Client[] clients, char objectType)
       throws ConnectionException, InvalidClientException,
              NoSuchSessionException, NoSuchChannelException,
              NoSuchByteArrayException, NoSuchClientException,
              NoSuchTokenException, PermissionDeniedException,
              TimedOutException;

public void
invite(Client[] clients, char objectType)
       throws ConnectionException, InvalidClientException,
              NoSuchSessionException, NoSuchChannelException,
```

```
                NoSuchByteArrayException, NoSuchClientException,
                NoSuchTokenException, PermissionDeniedException,
                TimedOutException;

public void
destroy(Client client, char objectType)
        throws ConnectionException, InvalidClientException,
                NoSuchSessionException, NoSuchChannelException,
                NoSuchByteArrayException, NoSuchClientException,
                NoSuchTokenException, PermissionDeniedException,
                TimedOutException;

public boolean
isManaged(char objectType, String objectName)
        throws ConnectionException, NoSuchSessionException,
                NoSuchChannelException, NoSuchByteArrayException,
                NoSuchTokenException, TimedOutException;

public void
join(Client client, boolean authenticate, char objectType)
        throws ConnectionException, InvalidClientexception,
                NoSuchByteArrayException, NoSuchChannelException,
                NoSuchClientException, NoSuchSessionException,
                NoSuchTokenException, PermissionDeniedException,
                NameInUseException, TimedOutException;

public void
leave(Client client, char objectType)
        throws ConnectionException, InvalidClientException,
                NoSuchByteArrayException, NoSuchChannelException,
                NoSuchClientException, NoSuchSessionException,
                NoSuchTokenException, TimedOutException;

public String[]
listClientNames(char objectType)
        throws ConnectionException,
                NoSuchSessionException, NoSuchChannelException,
                NoSuchByteArrayException, NoSuchTokenException,
                TimedOutException;
```

3.7 NamingProxy

The NamingProxy class contains the following methods that need to be completed for your implementation:

```
public void
initProxy(Hashtable connections, String host, int port)
        throws NoRegistryException, NoSuchHostException;


public Object
```

```
getProxy();

public void
bind(URLString urlString, Object object, Client client)
       throws AlreadyBoundException, ConnectionException,
              InvalidClientException, InvalidURLException,
              NoRegistryException, PermissionDeniedException,
              PortInUseException, TimedOutException;

public void
unbind(URLString urlString, Object object, Client client)
       throws ConnectionException, InvalidClientException,
              InvalidURLException, NoRegistryException,
              NotBoundException, PermissionDeniedException,
              TimedOutException;

public Object
lookup(URLString urlString)
       throws ConnectionException,
              NoRegistryException, InvalidURLException,
              NotBoundException, TimedOutException;

public void
addRegistryListener(RegistryListener listener)
       throws ConnectionException, NoRegistryException,
              NoSuchHostException, TimedOutException;

public void
removeRegistryListener(RegistryListener listener)
       throws ConnectionException, NoRegistryException,
              NoSuchHostException, NoSuchListenerException,
              TimedOutException;

public URLString[]
list()
       throws ConnectionException, NoRegistryException,
              TimedOutException;

public void
addConnectionListener(ConnectionListener listener);

public void
removeConnectionListener(ConnectionListener listener)
       throws NoSuchListenerException;
```

3.8 SessionProxy

The SessionProxy class contains the following methods that need to be completed for your implementation:

```
public void
```

```
initProxy(String name, SessionImpl session, Object object);

public Object
getProxy();

public void
attachSessionManager(SessionManager sessionManager,
                     Session session)
       throws ConnectionException,
              ManagerExistsException, NoSuchHostException,
              TimedOutException;

public ByteArray
createByteArray(Client client, String byteArrayName,
                byte[] value, int offset, int length,
                boolean autoJoin)
       throws ConnectionException, InvalidClientException,
              NameInUseException, NoSuchSessionException,
              NoSuchClientException, NoSuchHostException,
              PermissionDeniedException, TimedOutException;

public Channel
createChannel(Client client, String channelName,
                   boolean reliable, boolean ordered, boolean autoJoin)
       throws ConnectionException, InvalidClientException,
              NameInUseException, NoSuchSessionException,
              NoSuchClientException, NoSuchHostException,
              PermissionDeniedException, TimedOutException;

public Token
createToken(Client client, String tokenName, boolean autoJoin)
       throws ConnectionException, InvalidClientException,
              NameInUseException, NoSuchSessionException,
              NoSuchClientException, NoSuchHostException,
              PermissionDeniedException, TimedOutException;

public boolean
byteArrayExists(String byteArrayName)
       throws ConnectionException, NoSuchSessionException,
              TimedOutException;

public boolean
channelExists(String channelName)
       throws ConnectionException, NoSuchSessionException,
              TimedOutException;

public boolean
tokenExists(String tokenName)
       throws ConnectionException, NoSuchSessionException,
              TimedOutException;

public boolean
byteArrayManaged(String byteArrayName)
       throws ConnectionException, NoSuchByteArrayException,
```

```
                NoSuchSessionException, TimedOutException;

    public boolean
    channelManaged(String channelName)
           throws ConnectionException, NoSuchChannelException,
                  NoSuchSessionException, TimedOutException;

    public boolean
    tokenManaged(String tokenName)
           throws ConnectionException, NoTokenException,
                  NoSuchSessionException, TimedOutException;

    public ByteArray[]
    getByteArraysJoined(Client client)
           throws ConnectionException, InvalidClientException,
                  NoSuchSessionException, TimedOutException;

    public Channel[]
    getChannelsJoined(Client client)
           throws ConnectionException, InvalidClientException,
                  NoSuchSessionException, TimedOutException;

    public Token[]
    getTokensJoined(Client client)
           throws ConnectionException, InvalidClientException,
                  NoSuchSessionException, TimedOutException;

    public String[]
    listByteArrayNames()
           throws ConnectionException, NoSuchSessionException,
                  TimedOutException;

    public String[]
    listChannelNames()
           throws ConnectionException, NoSuchSessionException,
                  TimedOutException;

    public String[]
    listTokenNames()
           throws ConnectionException, NoSuchSessionException,
                  TimedOutException;

    public void
    close(boolean closeConnection)
           throws ConnectionException, NoSuchSessionException;
```

3.9 TokenProxy

The TokenProxy class contains the following methods that need to be completed for your implementation:

```
public void
initProxy(String name, SessionImpl session, Object object);

public Object
getProxy();

public int
give(Client client, String receivingClientName)
       throws ConnectionException, InvalidClientException,
              NoSuchTokenException, NoSuchClientException,
              NoSuchSessionException, PermissionDeniedException,
              TimedOutException;

public int
grab(Client client, boolean exclusive)
       throws ConnectionException, InvalidClientException,
              NoSuchTokenException, NoSuchClientException,
              NoSuchSessionException, PermissionDeniedException,
              TimedOutException;

public String[]
listHolderNames()
       throws ConnectionException, NoSuchSessionException,
              NoSuchTokenException, TimedOutException;

public int
request(Client client)
       throws ConnectionException, InvalidClientException,
              NoSuchTokenException, NoSuchClientException,
              NoSuchSessionException, PermissionDeniedException,
              TimedOutException;

public int
release(Client client)
      throws ConnectionException, InvalidClientException,
             NoSuchTokenException, NoSuchClientException,
             ClientNotGrabbingException, ClientNotReleasedException,
             NoSuchSessionException, PermissionDeniedException,
             TimedOutException;

public int
test() throws ConnectionException, NoSuchSessionException,
              NoSuchTokenException, TimedOutException;
```

3.10 Registry

The Registry class contains the following methods that need to be completed for your implementation:

```
public void
```

```
startRegistry(String registryType, int port)
     throws RegistryExistsException, NoRegistryException;

public void
stopRegistry(String registryType, int port)
     throws NoRegistryException;

public void
attachManager(RegistryManager manager)
     throws ManagerExistsException;

public boolean
registryExists(String registryType, int port)
     throws NoRegistryException;
```

3.11 <impl>Session

The <*impl*>Session class contains the following methods that need to be completed for your implementation:

```
public final synchronized void
_createProxy(NamingProxy namingProxy, String name, short sessionNo,
             String connectionType, String host, int port)
       throws NoSuchHostException;

public final synchronized void
_createServer(String name, short sessionNo, String connectionType,
              String url, int port);
```

3.12 <impl>Client

The <impl>Client class contains the following methods that need to be completed for your implementation:

```
public final Object
authenticate(AuthenticationInfo info);

public final String
getName();

public final void
byteArrayInvited(ClientEvent event);

public final void
byteArrayExpelled(ClientEvent event);

public final void
channelInvited(ClientEvent event);
```

```
public final void
channelExpelled(ClientEvent event);

public final void
sessionInvited(ClientEvent event);

public final void
sessionExpelled(ClientEvent event);

public final void
tokenInvited(ClientEvent event);

public final void
tokenExpelled(ClientEvent event);

public final synchronized void
_createProxy(NamingProxy namingProxy,
             String name, String host, int port)
      throws NoSuchHostException;

public final synchronized void
_createServer(String name, String url, int port);
```