Table of Contents

Writing a Source 2

Programmers Reference, Custom channels, Source And Sink, Write Source, example

Writing a Source

Before writing a Source you need to decide the type of Source that will receive messages from your software system or a hardware device. Three types of Sources are supported by the `openadaptor` framework:

1. *Polling* - the Source of this type will periodically poll a resource looking for data and messages that need to be processed or sent, e.g. a database, a file system or queue-based middleware.
2. *Listening* - the Source of this type will wait for a connection, establish a connection and listen, e.g. a socket.
3. *Callback* - the Source of this type will register for events and receive call-backs, e.g. publish/subscribe middleware.

You must decide which is the most appropriate type of Source for your situation. Depending on which implementation you choose, you will need to implement different methods.

After you decide which Source type you will use follow the steps below:

1. Inherit your class implementing the Source from the `AbstractSimpleSource` class.
2. Override the init method to set the Source type and/or initialise the Source with some specific properties. The Source-specific properties would have been entered by the user in the Configuration Tool (see also section 4.3) The type of the Source is set using the following method:

   ```
   setSourceType(POLL_SOURCE); or
   setSourceType(LISTEN_SOURCE); or
   setSourceType(CALLBACK_SOURCE); or
   ```

3. Depending on the Source type override the following methods:
   1. For polling Source -

      ```
      public DataObject[] sourcePoll() throws IbafException
      ```

      This method should return an array of `DataObjects` (or `null` if there is no data to process at the moment).

   2. For listening Source -

      ```
      public void sourceListen()
      throws IbafException
      ```

      This method should call the following method when it has some `DataObjects` to process:

      ```
      sourceProcess(DataObject[] dobs)
      ```

   3. For call-back Source – You will need to implement the third party call-back method. This method should call the following method when it has some DataObject's to process:

      ```
      sourceProcess(DataObject[] dobs)
      ```

In the methods above you need to construct the appropriate `DataObjects`, which will be delivered to ***Aware IM*** by the framework. To construct the DataObject's you can use the MessageBuilder class as shown in the example below.

4. You should implement the following method to perform any necessary "start up", such as establishing connections, registering call-backs, etc:

```
 public void sourceStartUp()
throws IbafException
```

5. You should implement the following method to perform necessary "clean up", such as destroying connections, releasing resources, etc:

```
public void sourceCleanUp()
throws IbafException
```

As an example you can see the code snippet that implements the Source that handles e-mails coming to a mailbox and sends them as notifications to ***Aware IM***. Note that this code is for illustration purposes only and certain details are omitted.

```
public  class EmailSource extends AbstractSimpleSource
{
    private Stringm_mailHost;
    private Stringm_userName;
    private Stringm_password;
    private Sessionm_session;
    private Store m_store;
    private Folder m_inbox;

/** Initializes the source.
* Mandatoryproperties
* MAIL_HOST_PROPnameofthemailhost,e.g.mail.optus.com
* MAIL_USER_PROPnameoftheuser
* MAIL_PASSWORD_PROPcredentialsoftheuser
* @exception IbafExceptionIfinitializationfails.
*/

public  void init(Stringname,Propertiesprops,Stringprefix,
Controllercontroller)
throws IbafException
{
    try
    {
        super .init(name,props,prefix,controller);
        // set the type of the source!
        setSourceType(CALLBACK_SOURCE);
        // get different settings that would have been entered//
        // by the user in the Configuration Tool
        m_mailHost=props.getProperty(prefix+"."+MAIL_HOST);
        if (m_mailHost== null )
            throw new IbafException("Mail host property for incoming e-mails
```

```
is not specified.");
        m_userName=props.getProperty(prefix+"."+MAIL_USER);
        if (m_userName== null )
            throw  new IbafException("User name property is not specified");
        m_password=props.getProperty(prefix+"."+ MAIL_PASSWORD);
        if (m_password== null )
            throw new IbafException("Password property is not specified");
        Propertiessettings=System.getProperties();
        settings.put("mail.smtp.host",m_mailHost);
        settings.put("mail.store.protocol","pop3");
        m_session=Session.getInstance(settings, null );
        m_store=m_session.getStore();
    }
    catch (Exceptione)
    {
        e.printStackTrace();
        throw new IbafException(e.getMessage());
    }
}

public  void sourceStartUp()
throws IbafException
{
    super .sourceStartUp();
    // try to connect to the remote host
    try
    {
        connect();
    }
    catch (Exceptionme)
    {
    }
}

public  void sourceCleanUp()
throws IbafException
{
    super .sourceCleanUp();
    // close everything
    if (m_inbox!= null )
    {
        synchronized (m_inbox)
        {
            try
            {
                m_inbox.close( false );
            }
            catch (Exceptione)
            {
            }
        }
```

```
    }
    if (m_store!= null )
    {
        try
        {
            m_store.close();
        }
        catch (Exceptione)
        {
        }
    }
}
/**  connect to the remote host and register our callback function */

private  void connect() throws Exception
{
    try
    {
        m_store.connect(m_mailHost, 110,m_userName, m_password);
    }
    catch (Exceptionse)
    {
    }
    // Open a Folder
    try
    {
        m_inbox=m_store.getFolder("INBOX");
    }
    catch (Exceptione)
    {
        throw  new IbafException(e.getMessage());
    }
    // Register our call back method with the third party API!!
    synchronized (m_inbox)
    {
        m_inbox.addMessageCountListener( new MessageCountAdapter()
        {
            public  void messagesAdded(MessageCountEventev)
            {
                gotMessage(ev);
            }
        });
    }
}

// This is a call-back method that creates data objects */
private  void gotMessage(Messagem)
{
    // create notification of a particular structure
INotificationnotif=DomainFactory.createNotification(DomainFactory.createInco
mingEmailNotification());
```

```
    // set attribute values
    try
    {
        notif.setAttributeValue(INCOMING_EMAIL_MAIL_HOST, m_mailHost);
        notif.setAttributeValue(INCOMING_EMAIL_USER_NAME, m_userName);
        // etc…
        // using MessageBuilder to construct required data
        // objects
        MessageBuildermb = new MessageBuilder();
        DataObject[]dobs=mb.addNotification( null ,notif);
        // send the data objects to Aware IM!
        sourceProcess(dobs);
    }
    catch (Exceptione)
    {
        // some error handling
    }
}
```

From:
http://www.awareim.com/dokuwiki/ - **Documentation**

Permanent link:
http://www.awareim.com/dokuwiki/docs/3500/0400/0410/0412?rev=1680756173

Last update: **2023/04/06 04:42**