Note – Make sure the `delete` option is set in the `server.policy` file, or expired file-based sessions might not be deleted properly. For more information about `server.policy`, see "The `server.policy` File" on page 56.

The Session Store Location setting also determines where the session state is persisted if it is not highly available; see "Choosing a Persistence Store" on page 155.

Stateful Session Bean Failover

An SFSB's state can be saved in a persistent store in case a server instance fails. The state of an SFSB is saved to the persistent store at predefined points in its life cycle. This is called *checkpointing*. If SFSB checkpointing is enabled, checkpointing generally occurs after any transaction involving the SFSB is completed, even if the transaction rolls back.

However, if an SFSB participates in a bean-managed transaction, the transaction might be committed in the middle of the execution of a bean method. Since the bean's state might be undergoing transition as a result of the method invocation, this is not an appropriate instant to checkpoint the bean's state. In this case, the EJB container checkpoints the bean's state at the end of the corresponding method, provided the bean is not in the scope of another transaction when that method ends. If a bean-managed transaction spans across multiple methods, checkpointing is delayed until there is no active transaction at the end of a subsequent method.

The state of an SFSB is not necessarily transactional and might be significantly modified as a result of non-transactional business methods. If this is the case for an SFSB, you can specify a list of checkpointed methods. If SFSB checkpointing is enabled, checkpointing occurs after any checkpointed methods are completed.

The following table lists the types of references that SFSB failover supports. All objects bound into an SFSB must be one of the supported types. In the table, *No* indicates that failover for the object type might not work in all cases and that no failover support is provided. However, failover might work in some cases for that object type. For example, failover might work because the class implementing that type is serializable.

TABLE 8–1 Object Types Supported for Java EE Stateful Session Bean State Failover

Java Object Type	Failover Support
Colocated or distributed stateless session, stateful session, or entity bean reference	Yes
JNDI context	Yes, `InitialContext` and `java:comp/env`
UserTransaction	Yes, but if the instance that fails is never restarted, any prepared global transactions are lost and might not be correctly rolled back or committed.

TABLE 8–1 Object Types Supported for Java EE Stateful Session Bean State Failover *(Continued)*

Java Object Type	Failover Support
JDBC DataSource	No
Java Message Service (JMS) ConnectionFactory, Destination	No
JavaMail Session	No
Connection Factory	No
Administered Object	No
Web service reference	No
Serializable Java types	Yes
Extended persistence context	No

For more information about the `InitialContext`, see "Accessing the Naming Context" on page 253. For more information about transaction recovery, see Chapter 14, "Using the Transaction Service." For more information about Administered Objects, see "Administering JMS Physical Destinations" in *GlassFish Server Open Source Edition 3.1 Administration Guide*.

Note – Idempotent URLs are supported along the HTTP path, but not the RMI-IIOP path. For more information, see "Configuring Idempotent URL Requests" on page 136.

If a server instance to which an RMI-IIOP client request is sent crashes during the request processing (before the response is prepared and sent back to the client), an error is sent to the client. The client must retry the request explicitly. When the client retries the request, the request is sent to another server instance in the cluster, which retrieves session state information for this client.

HTTP sessions can also be saved in a persistent store in case a server instance fails. In addition, if a distributable web application references an SFSB, and the web application's session fails over, the EJB reference is also failed over. For more information, see "Distributed Sessions and Persistence" on page 114.

If an SFSB that uses session persistence is undeployed while the GlassFish Server instance is stopped, the session data in the persistence store might not be cleared. To prevent this, undeploy the SFSB while the GlassFish Server instance is running.

Configure SFSB failover by:

- "Choosing a Persistence Store" on page 155
- "Enabling Checkpointing" on page 156
- "Specifying Methods to Be Checkpointed" on page 157

Choosing a Persistence Store

The following types of persistent storage are supported for passivation and checkpointing of the SFSB state:

- **The local file system** - Allows a single server instance to recover the SFSB state after a failure and restart. This store also provides passivation and activation of the state to help control the amount of memory used. This option is not supported in a production environment that requires SFSB state persistence. This is the default storage mechanism if availability is *not* enabled.
- **Other servers** - Uses other server instances in the cluster for session persistence. Clustered server instances replicate session state. Each backup instance stores the replicated data in memory. This is the default storage mechanism if availability is enabled.

Choose the persistence store in one of the following ways:

- To use the local file system, first disable availability. Select the Availability Service component under the relevant configuration in the Administration Console. Uncheck the Availability Service box. Then select the EJB Container component and edit the Session Store Location value. The default is *domain-dir*`/session-store`.
- To use other servers, select the Availability Service component under the relevant configuration in the Administration Console. Check the Availability Service box. To enable availability for the EJB container, select the EJB Container Availability tab, then check the Availability Service box. All instances in an GlassFish Server cluster should have the same availability settings to ensure consistent behavior.

For more information about SFSB state persistence, see the *GlassFish Server Open Source Edition 3.1 High Availability Administration Guide*.

Using the --keepstate Option

If you are using the file system for persistence, you can use the `--keepstate` option of the `asadmin redeploy` command to retain the SFSB state between redeployments.

The default for `--keepstate` is false. This option is supported only on the default server instance, named `server`. It is not supported and ignored for any other target.

Some changes to an application between redeployments prevent this feature from working properly. For example, do not change the set of instance variables in the SFSB bean class.

If any active SFSB instance fails to be preserved or restored, *none* of the SFSB instances will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.

To preserve active state data, GlassFish Server serializes the data and saves it in memory. To restore the data, the class loader of the newly redeployed application deserializes the data that was previously saved.

For more information about the `asadmin redeploy` command, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Using the --asyncreplication Option

If you are using replication on other servers for persistence, you can use the `--asyncreplication` option of the `asadmin deploy` command to specify that SFSB states are first buffered and then replicated using a separate asynchronous thread. If `--asyncreplication` is set to true (default), performance is improved but availability is reduced. If the instance where states are buffered but not yet replicated fails, the states are lost. If set to false, performance is reduced but availability is guaranteed. States are not buffered but immediately transmitted to other instances in the cluster.

For more information about the `asadmin deploy` command, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Enabling Checkpointing

The following sections describe how to enable SFSB checkpointing:

- "Server Instance and EJB Container Levels" on page 156
- "Application and EJB Module Levels" on page 156
- "SFSB Level" on page 156

Server Instance and EJB Container Levels

To enable SFSB checkpointing at the server instance or EJB container level, see "Choosing a Persistence Store" on page 155.

Application and EJB Module Levels

To enable SFSB checkpointing at the application or EJB module level during deployment, use the `asadmin deploy` or `asadmin deploydir` command with the `--availabilityenabled` option set to `true`. For details, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

SFSB Level

To enable SFSB checkpointing at the SFSB level, set `availability-enabled="true"` in the `ejb` element of the SFSB's `glassfish-ejb-jar.xml` file as follows:

```
<glassfish-ejb-jar>
   ...
   <enterprise-beans>
      ...
      <ejb availability-enabled="true">
```

```
         <ejb-name>MySFSB</ejb-name>
      </ejb>
   ...
   </enterprise-beans>
</glassfish-ejb-jar>
```

Specifying Methods to Be Checkpointed

If SFSB checkpointing is enabled, checkpointing generally occurs after any transaction involving the SFSB is completed, even if the transaction rolls back.

To specify additional optional checkpointing of SFSBs at the end of non-transactional business methods that cause important modifications to the bean's state, use the `checkpoint-at-end-of-method` element within the `ejb` element in `glassfish-ejb-jar.xml`.

For example:

```
<glassfish-ejb-jar>
   ...
   <enterprise-beans>
      ...
      <ejb availability-enabled="true">
         <ejb-name>ShoppingCartEJB</ejb-name>
         <checkpoint-at-end-of-method>
            <method>
               <method-name>addToCart</method-name>
            </method>
         </checkpoint-at-end-of-method>
      </ejb>
      ...
   </enterprise-beans>
</glassfish-ejb-jar>
```

For details, see "checkpoint-at-end-of-method" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

The non-transactional methods in the `checkpoint-at-end-of-method` element can be the following:

- `create` methods defined in the home or business interface of the SFSB, if you want to checkpoint the initial state of the SFSB immediately after creation
- For SFSBs using container managed transactions only, methods in the remote interface of the bean marked with the transaction attribute TX_NOT_SUPPORTED or TX_NEVER
- For SFSBs using bean managed transactions only, methods in which a bean managed transaction is neither started nor committed

Any other methods mentioned in this list are ignored. At the end of invocation of each of these methods, the EJB container saves the state of the SFSB to persistent store.

Note – If an SFSB does not participate in any transaction, and if none of its methods are explicitly specified in the `checkpoint-at-end-of-method` element, the bean's state is not checkpointed at all even if `availability-enabled="true"` for this bean.

For better performance, specify a *small* subset of methods. The methods chosen should accomplish a significant amount of work in the context of the Java EE application or should result in some important modification to the bean's state.

Session Bean Restrictions and Optimizations

This section discusses restrictions on developing session beans and provides some optimization guidelines.

- "Optimizing Session Bean Performance" on page 158
- "Restricting Transactions" on page 158

Optimizing Session Bean Performance

For stateful session beans, colocating the stateful beans with their clients so that the client and bean are executing in the same process address space improves performance.

Restricting Transactions

The following restrictions on transactions are enforced by the container and must be observed as session beans are developed:

- A session bean can participate in, at most, a single transaction at a time.
- If a session bean is participating in a transaction, a client cannot invoke a method on the bean such that the `trans-attribute` element (or `@TransactionAttribute` annotation) in the `ejb-jar.xml` file would cause the container to execute the method in a different or unspecified transaction context or an exception is thrown.
- If a session bean instance is participating in a transaction, a client cannot invoke the `remove` method on the session object's home or business interface object, or an exception is thrown.

Using Read-Only Beans

A *read-only bean* is an EJB 2.1 entity bean that is never modified by an EJB client. The data that a read-only bean represents can be updated externally by other enterprise beans, or by other means, such as direct database updates.