Contents

The *EntityManager* can serialize or **save its cache** to a stream or file.

You may want to save cached entities when:

- The application must be able to run offline for extended periods, and you need to preserve the state of the cache if the application or computer is shut down.
- The user may accumulate many changes over a long time and you need to backup their data. This may also occur because the user is not yet ready to commit their changes to the datastore, or because the modifications to the entities are incomplete, and they might not pass validation checks required for saving.

Often, especially in Silverlight applications, isolated storage is used to hold the saved data, but any Stream can be used. For WinClient applications, saving directly to a file is also available.

Save and restore

To access the EntityManager's cache, use its *CacheStateManager* property. The *CacheStateManager* class contains the methods *SaveCacheState* and *RestoreCacheState* which are used to save and load the cache.

Using *SaveCacheState*, you can save to a stream or file either the entire *EntityManager* cache or only a subset of entities.

With *RestoreCacheState*, you can restore a previously saved cache state into an *EntityManager*. When restoring you can also determine the *RestoreStrategy* to use. The *RestoreStrategy* determines how the entities are merged if they already exist in the EntityManager's cache and whether other default settings on the EntityManager should be restored.

If you are restoring the cache into an empty EntityManager, then you don't need to worry about merge conflicts. This only occurs in advanced scenarios where you want to merge the incoming cache into an already active EntityManager that may have changes you want to preserve. (For example, you are importing another user's cache, or you are restoring a backup cache, but want to keep the current modifications.)

In most cases, a RestoreStrategy of *Normal* is used. This preserves all of the changes in the *EntityManager* so that they take precedence in the event of a conflict with the cache being imported. However, you can configure your own *RestoreStrategy* if needed. For an in depth discussion about merging entities see Merge query results into the entity cache.

Both *SaveCacheState* and *RestoreCacheState* also allow you to save/restore data in text format. Binary format is used by default, but text format can be useful when diagnosing problems or for situations when the serialized data will be used by non-DevForce applications.

Sample save/restore using Isolated Storage

```
private void SaveCacheToIsolatedStorage(EntityManager manager, String fileName) {
 // Remember to add error handling!
 using (var isoFile = IsolatedStorageFile.GetUserStoreForApplication()) {
  using (var isoStream = new IsolatedStorageFileStream(fileName, FileMode.Create, isoFile)) {
    manager.CacheStateManager.SaveCacheState(isoStream);
   }
  }
}
private void RestoreCacheFromIsolatedStorage(EntityManager manager, String fileName) {
 // Remember to add error handling!
 using (var isoFile = IsolatedStorageFile.GetUserStoreForApplication()) {
  using (var isoStream = new IsolatedStorageFileStream(fileName, FileMode.Open, isoFile)) {
    manager.CacheStateManager.RestoreCacheState(isoStream, RestoreStrategy.Normal);
   }
  }
}
```

```
Private Sub SaveCacheToIsolatedStorage(ByVal manager As _
  EntityManager, ByVal fileName As String)
 ' Remember to add error handling!
 Using isoFile = IsolatedStorageFile.GetUserStoreForApplication()
  Using isoStream = New IsolatedStorageFileStream(fileName, FileMode.Create, isoFile)
   manager.CacheStateManager.SaveCacheState(isoStream)
  End Using
 End Using
```

```
End Sub
Private Sub RestoreCacheFromIsolatedStorage(ByVal manager As _
  EntityManager, ByVal fileName As String)
 ' Remember to add error handling!
 Using isoFile = IsolatedStorageFile.GetUserStoreForApplication()
   Using isoStream = New IsolatedStorageFileStream(fileName, FileMode.Open, isoFile)
     manager.CacheStateManager.RestoreCacheState(isoStream, RestoreStrategy.Normal)
   End Using
 End Using
End Sub
```

Save custom properties

If you have defined custom properties on your entity (properties that are not backed by the datastore and are not calculated) and want these serialized along with the entity, you should mark them with the *DataMember* attribute and implement a public get and set for the property. (Remember to include a reference to the System.Runtime.Serialization assembly.)

```
[DataMember]
public String Note { get; set; }
```

```
<DataMember>
Public Property Note() As String
```

In addition, by marking the property with the *DataMember* attribute, the value will also serialize between the client and *EntityServer*, so if you set it in one location, it will be available in the other. You can also serialize complex types.

Other technical considerations

Note that the temporary id map is also serialized to the stream. This allows you to create new entities without the server (even when connected), and for id fixup to be performed on the entity graph when the entities are committed to the database. See Custom id generation to learn more.

When restoring the entity cache, the query cache is also cleared. See Cache a query to learn more.

The techniques described here can also be used to support both design time data and testing.