Entities can be **serialized into an XML stream or file** for any number of purposes, including exposing these entities from a Web Service, or as the first step in an XSLT transform for reporting or further processing.

All entities in the code generated by the Entity Data Model Designer are marked with the DataContract attribute, and all entity properties with the DataMember attribute. These attributes define how an entity will be serialized by a serializer. They are also required by DevForce in an n-tier application to query and save entities across tiers.

Serialization using the DataContractSerializer and NetDataContractSerializer is supported.

Serialization using both DataContractJsonSerializer and JavaScriptSerializer are not supported. These are known issues acknowledged by Microsoft. The .NET community recommends using Json.NET instead.

Depth of the object graph

One issue when serializing an object graph (objects that are connected to other objects, ad-infinitum) has to do with the depth of the object graph that should be serialized. Navigation properties on entities in DevForce are lazily loaded, so you might at first worry that serialization of a single entity could accidentally cause a large object graph to be serialized. Fortunately, DevForce controls this by only serializing entities that are present within the *EntityManager's* cache at the start of serialization. No queries will be sent to the data source to load navigation properties not yet loaded, so only the data you want will be serialized.

For example, if you load an Employee and her Orders into the *EntityManager* cache and then serialize the Employee object, the result will also include the loaded Orders. If LineItems for the Orders had also been loaded into cache, then the object graph serialized would include them too.

Alternately, if only an Employee was in cache without any attached Orders, then only the Employee object would be serialized.

The *EntityCacheState*

Some applications may find the built-in EntityCacheState an easy means of serializing and deserializing entities. The *EntityCacheState* is a serializable snapshot of the *EntityManager's* cache, or of a selected subset of entities. The *EntityCacheState* can be serialized in text or binary format, and can be transmitted across tiers via a remote server method.