Workflow for a save

During the execution of a save a number of discrete steps are followed. DevForce provides several points along the way where a developer can intervene and modify the execution path. This page describes this sequence of steps and the interception points.

Component	Action
Client Tier **Application Code**	The client application adds, modifies and deletes any number of business objects on the client. The client application asks a **EntityManager** to save all pending changes.
Client Tier **EntityManager**	Makes a save list of all of the new, modified, and deleted entities in the EntityManager entity cache. Fires the *Saving* event, passing in the list of entities to be saved. The listener can add or remove entities from this list or cancel the save. For now, let's assume that application listener just okays the save. Transmits the list of entities to be saved to the *EntityServer*.
Middle Tier **EntityServer**	Server authenticates the user. Let's assume success. The EntityServer creates a new *EntityServerSaveInterceptor* or an instance of a developer customized subclass. In the *EntityServerSaveInterceptor* the *AuthorizeSave* method can be used to perform security checks; the *ValidateSave* method can be used to perform server-side validation. When the *EntityServerInterceptor.ExecuteSave* method is called, where DevForce performs a temporary id to permanant id conversion and then forwards the saves to the Entity Framework for execution.
Data Tier **Data Source**	Performs the persistence operations. If there are no failures, it commits them; if there is a single failure, it rolls them all back.
Middle Tier – EntityServer	If the transaction failed, returns to the *EntityManager* the identity of the culprit entity and the exception raised by the data source. The *EntityManager* stores this information in the *SaveResult* and returns to the client application. Workflow ends. Otherwise… The transaction succeeded. The *EntityServer* re-queries the database(s) for all of the inserted and modified entities that are sourced in databases, thus capturing the effects of triggers that fired during save. Converts the (potentially) revised data into entities and sends them to the client side *EntityManager*. The server's local copy of the entities go out of scope and the garbage collector reclaims them. This enables the object server to stay stateless.
Client Tier **EntityManager**	Performs id fixup, converting temporary ids to permanent ids based on a mapping between the two generated by the EntityServer during the save and passed back to the client in the previous step. Replaces cached entities with updates from *EntityServer*. They are marked "unchanged" because they are now current. Raises the *Saved* event with list of saved inserted and modified entities.
Client Tier **Application Code**	The application resumes.