Persistence Framework  
Technical Design Document  
Version 2.7

Table of Contents

[Persistence Framework 3](#_Toc201302485)

[‘PersistenceObject’ Data Object 3](#_Toc201302486)

[‘PersistenceAttribute’ Data Object 4](#_Toc201302487)

[‘Persistence Object Mapping Wrapper’ Service Flow 5](#_Toc201302488)

[‘Persistence Object Mapping’ Service Flow 5](#_Toc201302489)

[‘Persistence Object Marshalling’ Service Flow 5](#_Toc201302490)

[‘Save’ Service Flows 6](#_Toc201302491)

[‘Load’ Service Flows 6](#_Toc201302492)

[Date and Time Considerations 7](#_Toc201302493)

[Performance Tips 7](#_Toc201302494)

[Known Limitations 7](#_Toc201302495)

# Persistence Framework

The **Persistence Framework Toolkit (PERSIST)** was designed to allow for easily mapping BAW data objects to their underlying SOR database tables and columns via a simple, standardized, and extensible configuration and rapidly building out Load and Save database services to retrieve and persist those objects, respectively. The framework consists of the assets described in the remainder of this section.

## ‘PersistenceObject’ Data Object

This BAW object is used to define a specific BAW data object that supports persistence in the process application. The various required attributes and supported optional attributes are described below:

**entityName** – the required name of the BAW data object.

**entityType** – the required type of entity, either ‘ReferenceEntity’ or ‘InstanceEntity’ from the **PersistenceEntityType** localization resource. A ‘ReferenceEntity’ is used to represent an entity that serves as master data or configuration data such as a **Status** object which would be pre-loaded to the SOR database via a SQL script. Meanwhile, an ‘InstanceEntity’ would be data specific to a user- or system-created object, such as a **Request** or **Comment**.

**tableName** – the required database table name that stores the data for the given **entityName**.

**sequenceName** – while technically an optional value, this is required for InstanceEntity object types and tells the system which sequence should be used to generate the next ID from.

**primaryKeyAttributeName** – this required value is the name of the attribute that corresponds to the primary key for the object, typically **id**.

**defaultOrderBy** – this optional value determines the sort order used for selected rows of this object type from Load services.

**persistenceAttributes** – this required list of objects defines specifics about each of the attributes of the object and how they are to be persisted or loaded. More details about these are found below in the **PersistenceAttribute** object description.

**primaryKeyPersistenceAttribute** – this is an attribute used by the persistence framework services to simplify some processing and is not intended to be referenced or defined by a developer.

**selectAliasMap** – this is an attribute used by the persistence framework services to simplify some processing and is not intended to be referenced or defined by a developer.

**typeMap** – this is an attribute used by the persistence framework services to simplify some processing and is not intended to be referenced or defined by a developer.

**nestedObjectPrimaryKeyMap** – this is an attribute used by the persistence framework services to simplify some processing and is not intended to be referenced or defined by a developer.

**whereAliasMap** – this is an attribute populated by the persistence framework services to allow the developer to reference the appropriate JOIN tables when WHERE criteria includes filters on columns in the foreign table. This is typically only referenced in the object’s Load service when meeting the above criteria.

## ‘PersistenceAttribute’ Data Object

This BAW object is used to define the attributes of a **PersistenceObject**, as well as providing some optional configuration values to accommodate different data scenarios that needed to be accounted for, such as entity relationships.

**attributeName** – this required value is the name of the BAW attribute.

**columnName** – this required value is the name of the database column that corresponds to the particular attribute.

**databaseType** – this required value is the data type for the database column – typically ‘VARCHAR’, ‘INTEGER’, ‘DECIMAL’, ‘DATE’, etc.

**nestedEntityName** – this optional attribute is supplied when the attribute is a nested object, and the nested object’s entity name should be supplied.

**nestedPersistenceObject** – this is an attribute used by the persistence framework services to simplify some processing and is not intended to be referenced or defined by a developer.

**nestedLevels** – this optional attribute allows the developer to define the maximum number of levels of nested recursion that should be processed. This is necessary for situations with recursively nested objects, which may contain a list of the same object. Without this setting, the persistence services would continue to process the nested objects in an infinite loop. The default for this value is an environment variable named **PERSISTENCE\_MAX\_NESTED\_LEVELS** and is set to 5.

**relationshipTable** – this optional attribute is used when the attribute is a nested object and specifically joined to the parent object in a many-to-many relationship via a relationship table. The name of the relationship table should be supplied here.

**relationshipParentColumn** – this optional attribute should be used in conjunction with the **relationshipTable** attribute and would specify the name of the column in the relationship table that should be joined to the parent object’s primary key.

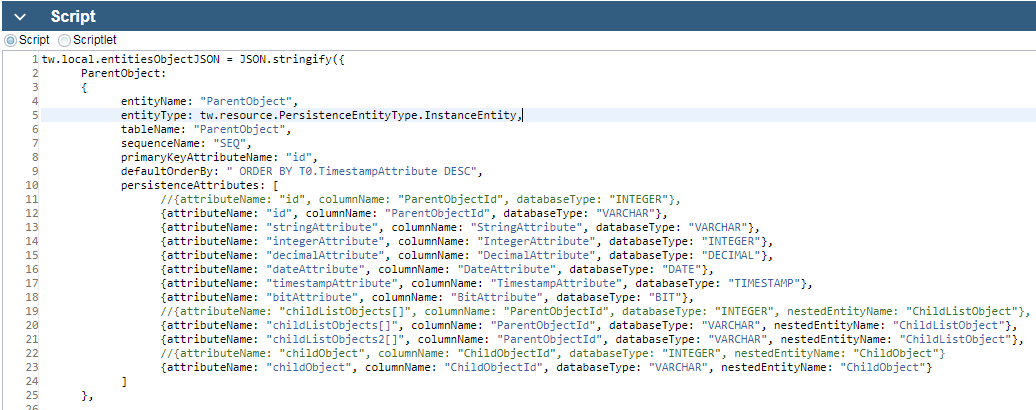
**relationshipChildColumn** – this optional attribute should be used in conjunction with the **relationshipTable** attribute and would specify the name of the column in the relationship table that should be joined to the child object’s primary key.

**forceZeroToNull** – this optional attribute is intended to be used to address the BAW behavior where an Integer attribute (think of the **id**) of an optional nested object will automatically be initialized from **null** to 0 after being bound to a coach or coachview. This check is made during the INSERT/UPDATE logic and will change the value from 0 back to **null** prior to a Save operation.

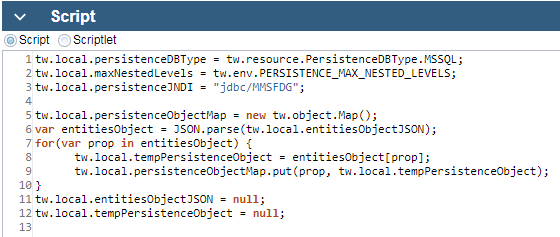
**preventUpdates** – this optional attribute is intended for (**and currently only implemented for**) lists of nested **InstanceEntity** objects and identifies the **PersistenceAttribute** as one that cannot be updated or deleted by the PersistenceFramework. This is useful for entities such as comments, task history, and/or audit trails where the data should allow additional items to be added to the list, but no items in the list should ever be updated after they are saved. This prevents additional UPDATE statements from being executed on child objects and can yield significant performance gains when used on large lists of child objects that do not require updates.

## ‘Persistence Object Mapping Wrapper’ Service Flow

This service contains the mapping between a BAW business object and the corresponding database details. Specifically, the **JS Object Mapping** script block should contain an entry similar to the following for each BAW business object that needs to be persisted or loaded from the SOR:



Additionally, the **Persistence Settings** script block should define the following settings for the process application’s persistence:



**IMPORTANT NOTE:** This service is just a sample wrapper included in the toolkit, but upon implementing the Persistence Toolkit in a process application a copy of this service should be made in the corresponding process app with a name like {**ProcessAppAcronym} Persistence Object Mapping Wrapper**. This is necessary to define the process-app specific settings for the framework.

## ‘Persistence Object Mapping’ Service Flow

This uses the provided **persistenceObjectMap** variable to generate any required sequences for new entities in **Save** services and also is responsible for generating the corresponding SELECT or INSERT/UPDATE SQL commands to persist the object.

## ‘Persistence Object Marshalling’ Service Flow

The job of this service is to process the results returned from a corresponding Save or Load operation and transform or marshall those results into the resulting object or object lists based on the **PersistenceObject** and **PersistenceAttribute** definitions and any configuration options provided.

**IMPORTANT NOTE:** This service is just a sample wrapper included in the toolkit, but upon implementing the Persistence Toolkit in a process application a copy of this service should be made in the corresponding process app with a name like {**ProcessAppAcronym} Persistence Object Marshalling Wrapper**. The reason this service must be copied to the process app level is because the object definitions either exist within this process app or a dependent toolkit. If the service inside the **PERSIST** toolkit is referenced, converting the objects from JS objects to BAW objects will fail as the toolkit does not know about the BAW business objects.

## ‘Save’ Service Flows

To persist a new object to the SOR, the following steps are required:

1. Add the new object to the database ERD and create the corresponding table using SQL in the system of record database schema.
2. Create the new BAW business object with all attributes.
3. Define the object and database table configuration in the **{ProcessAppAcronym} Persistence Object Mapping Wrapper** service.
4. Create a new service flow named **{EntityName} Save** – copying or duplicating from an existing Save service (or from the toolkit’s **ParentObject Save** service flow) is the easiest way to do this as it includes all of the necessary components.
5. Change the input and output on the service to be named according to the new object type and set the actual object type accordingly.
6. Update the input and output values for the ‘entity’ parameter of type ANY on the **{ProcessAppAcronym} Persistence Object Mapping Wrapper** service flow.
7. Create a unit test wrapper or “test harness” to validate that the new Save service works and all underlying SQL, objects, configuration, and services are in place. It is also recommended that this test harness be copied from an existing service flow that includes the pattern for building a sample object to persist using XML or JSON.

## ‘Load’ Service Flows

To load a new object from the SOR, the following steps are required:

1. Add the new object to the database ERD and create the corresponding table using SQL in the system of record database schema.
2. Create the new BAW data object with all attributes.
3. Define the object and database table configuration in the **{ProcessAppAcronym} Persistence Object Mapping Wrapper** service.
4. Create a new service flow named **{EntityName} Load** – copying or duplicating from an existing Get service is the easiest way to do this as it includes all of the necessary components.
5. Remove all inputs to the service except for any lookup or filter values that are needed for this object.
6. The outputs should be updated to be named after the object type and should output a list of that object type. If situations may arise where a single object of this type may frequently need to be loaded, i.e. using a ‘byId’ input, then a singular output object may also exist.
7. Update the **entityName** input on the **{ProcessAppAcronym} Persistence Object Mapping Wrapper** service flow to match the name of this object type – this is a rare case where the string can be hardcoded.
8. Update the **WHERE** script block using the existing patterns on the service (or referenced from other Load services) to construct any WHERE conditions needed to apply the proper filter inputs to the database records to return the desired result set.
9. Finally, update the output parameter mapping on the **{ProcessAppAcronym} Persistence Object Marshalling Wrapper** service to match the service flow’s output list of objects.
10. Finally, if a single object is also output, that can be set via a **Post** script on the **{ProcessAppAcronym} Persistence Object Marshalling Wrapper** service flow that follows this pattern:

if(tw.local.<objects> && tw.local.<objects>[0])  
 tw.local.<object> = tw.local.<objects>[0];

1. Create a unit test wrapper or “test harness” to validate that the new Load service works for all filter inputs and confirm that all underlying SQL, objects, configuration, and services are in place.

**IMPORTANT NOTE:** The **Persistence Object Mapping** service provides an input for **excludeProperties (List of String)** that should be mapped as an input to the **Persistence Object Mapping Wrapper** service. This option allows the developer to exclude certain attributes from loading, which can have huge performance gain implications. Consider a scenario where an object needs to be loaded and may contain a nested lists of taskHistory[] and comments[] objects. By excluding these attributes, the SQL JOINs necessary to return these data elements are eliminated saving on the SQL query execution time, and additionally there will be a lot fewer records in the results which can significantly save time in the Persistence Object Marshalling service as the objects are merged into the final output. This is a very important setting to understand for performance.

## Date and Time Considerations

It is important to note that all dates stored in the system of record using the **Persistence Framework Toolkit** are stored in UTC time – the idea behind this was to make the implementation consistent across the application and across timezones. The persistence framework takes care of converting all Date types to UTC prior to inserting or updating the values in the database, and conversely upon loading Date data out of the database it is converted from UTC to server time. The one place that developers need to keep this in mind is when writing queries against the SOR that include Date components, such as in the WHERE clause. The developer must remember to convert the time to UTC so that the correct condition is applied, otherwise there is a risk that the condition will be off by several hours (i.e. the number of hours from the server timezone offset to UTC time).

## Performance Tips

* Be sure to understand how the **preventUpdates** attribute works in the **PersistenceAttribute** object.
* Use the **excludeProperties** input on any **<Entity> Load** service flows to specify any nested lists of child objects that do not need to be loaded as this can drastically improve the performance of an **<Entity> Load** service.

## Known Limitations

* If a complex object has two attributes which are both lists of the same nested complex type, the framework currently has no way to differentiate between the lists so the all elements would be loaded into both lists.  
    
  Here is an example that would not work:
  1. Consider a process app that contains two custom complex objects, one called **Request** and one called **Address**.
  2. The **Request** object has two attributes which are both lists: **Request.oldAddresses[]** and **Request.newAddresses[]**
  3. Here is a sample Request object:

{

oldAddresses: [

{street: “123 Main St.”, city: “Anytown”, state: “AA”},

{street: “456 Test Dr.”, city: “Somewhere”, state: “ZZ”}

],

newAddresses: [

{street: “555 Mid Rd.”, city: “There”, state: “MM”},

]

}

* 1. When the following object is saved, all three child **Address** records will have a **RequestId** column in their database record linking it to the same **Request**.
  2. Therefore, when the **Request** is loaded, it will look like this:

{

oldAddresses: [

{street: “123 Main St.”, city: “Anytown”, state: “AA”},

{street: “456 Test Dr.”, city: “Somewhere”, state: “ZZ”},

{street: “555 Mid Rd.”, city: “There”, state: “MM”}

],

newAddresses: [

{street: “123 Main St.”, city: “Anytown”, state: “AA”},

{street: “456 Test Dr.”, city: “Somewhere”, state: “ZZ”},

{street: “555 Mid Rd.”, city: “There”, state: “MM”}

]

}

It turns out this is actually a common problem with persistence frameworks in general, noted by this Java JPA/Hibernate question of the same nature:

<https://stackoverflow.com/questions/669828/how-to-have-2-collections-of-the-same-type-in-jpa>

There does appear to be a way to resolve this by defining a relationship table – which the BAW Persistence Framework supports – however it would need to be a relationship between an **InstanceEntity** and another **InstanceEntity** which is not currently implemented because it has never been a necessary use case. The current relationship table pattern supports an **InstanceEntity** and a child **ReferenceEntity**, such as parent complex type with a list of reference objects. If supporting two nested lists of the same complex type is necessary, adjustments will need to be made to the Persistence Framework in order to support this.