Production Timer Web API

By

David Scott Blackburn

This document will discuss the how the back-end of the Production Timer is designed. The back-end of this project is accessed via the Production Timer Web API. The following will be explained in this document:

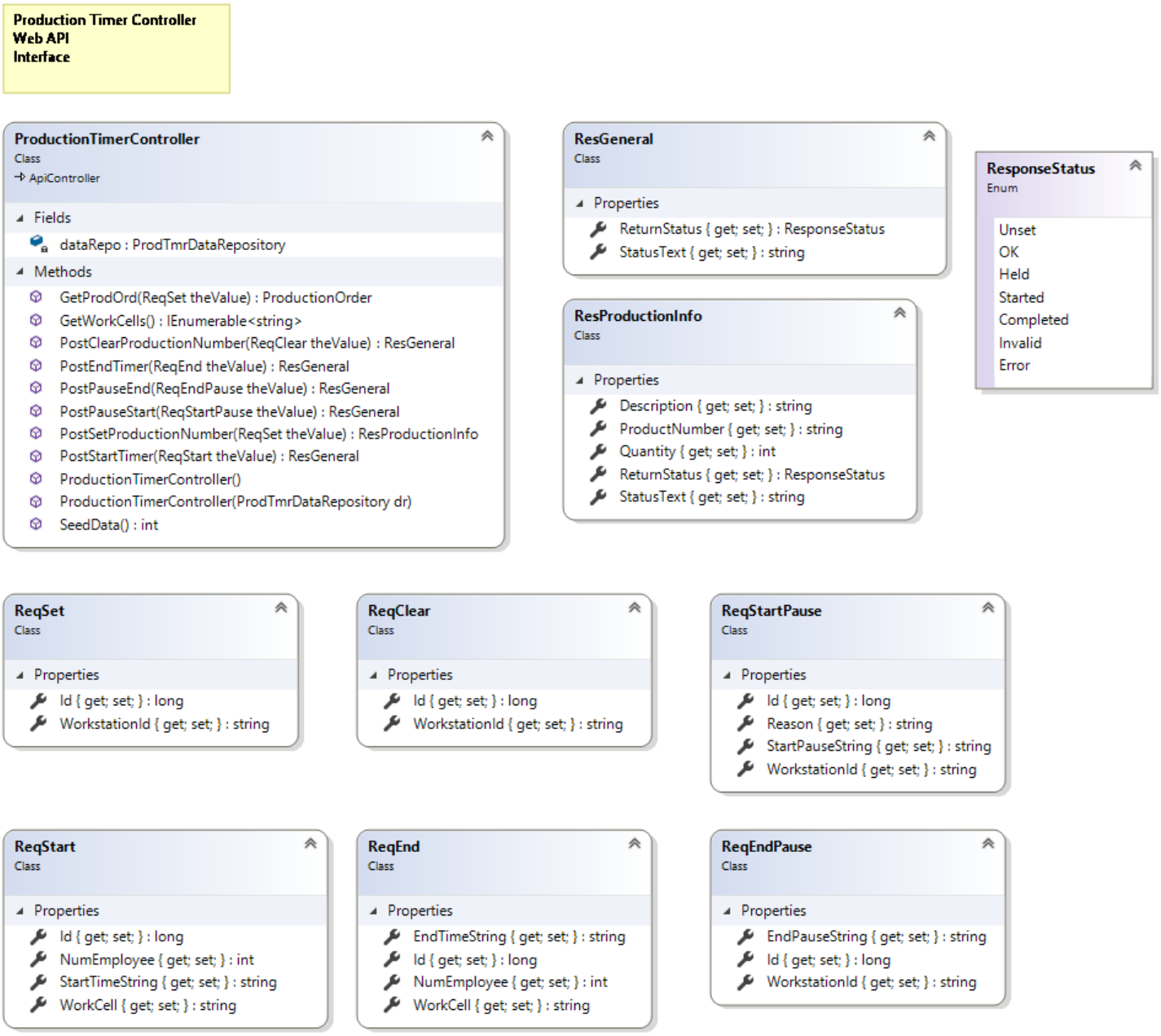
- Web API's RESTful Interface
- Interface Class structure
- Entity Framework and business model
- Database Structure

This program is an emulation of a much larger and sophisticated system. The structures used in this project do not match in anyway the actual structures used in the original system. The data structure used here is the one I would use based on the portion of the system I emulated and to possible project expansion.

RESTful Interface

Call Type	URI	Purpose	Request Data	Response Data
GET	api/ProductionTimer/WorkCells	Returns a list of Work Cells	None	Array of Strings
POST	api/ProductionTimer/Set	Sets Production Order status to SET and returns base Production Order information	ReqSet	ResProductionInfo
POST	api/ProductionTimer/Clear	Reset Production Order Status back to Ready	ReqClear	ResGeneral
POST	api/ProductionTimer/StartTimer	Sets status to Start and sets the start time	ReqStart	ResGeneral
POST	api/ProductionTimer/EndTimer	Sets status to Completed and set the End Time and the duration	ReqEnd	ResGeneral
POST	api/ProductionTimer/PauseStart	Creates a Start Pause record for either a Production Order (Detail) or to Pause Unassigned	ReqStartPause	ResGeneral
POST	api/ProductionTimer/PauseEnd	Sets the end time and duration for the previously started pause.	ReqEndPause	ResGeneral
POST	api/ProductionTimer/SeedData	Causes the data to be reset	None	None
POST	api/ProductionTimer/Get	Returns details for a particular Production Order	ReqSet	ProductionOrder

This is a simple Web API. In general, an action exists for each POST action. Data is converted, if needed, and a call made to the appropriate database repository method.



Interface Class structure

The above diagram shows the details regarding the Interface classes. The previous section explained the details regarding the REST interface that is how information is transferred to and from the Web API.

Each of the Production Timer Controller methods (actions) makes a call to the Work Cell Repository or Production Order Repository method.

Production Timer
Business Logic Data Model
and Entity Framework Model (from the Business Model side)

This model shows the Business Logic Model

This program primarily uses Stored Procedures or views to retrieve and update data in the database.

Refer to the Production Timer Controller Diagram and the Production Timer Data Diagram for further information.

IDisposable

ProdTmrDataRepository
Class

- Fields
  - entities : ProdTimerEntities
- Properties
  - ProductionOrder { get; set; } : IProductionOrder
  - WorkCellRepository { get; set; } : IWorkCell
- Methods
  - ~ProdTmrDataRepository()
  - Dispose() : void
  - Dispose(bool disposing) : void
  - ProdTmrDataRepository()
  - ProdTmrDataRepository(IProductionOrder po, IWorkCell wc)

IWorkCell
Interface

- Methods
  - *GetAllWorkCells() : List<WorkCell>*

IWorkCell

WorkCellRepository
Class

- Fields
  - entities : ProdTimerEntities
- Methods
  - GetAllWorkCells() : List<WorkCell>
  - WorkCellRepository(ProdTimerEntities entities)

WorkCell
Class

- Properties
  - id { get; set; } : int
  - name { get; set; } : string
  - workAreaId { get; set; } : int

IProductionOrder

IProductionOrder
Interface

- Methods
  - *ClearProductionId(ReqClear clearData) : ResGeneral*
  - *EndTimer(ReqEnd endData) : ResGeneral*
  - *GetProductionOrder(long id) : ProductionOrder*
  - *PauseEnd(ReqEndPause endPauseData) : ResGeneral*
  - *PauseStart(ReqStartPause startPauseData) : ResGeneral*
  - *SeedData() : void*
  - *SetProductionId(ReqSet setData) : ResProductionInfo*
  - *StartTimer(ReqStart startData) : ResGeneral*

ProductionOrderRepository
Class

- Fields
  - entities : ProdTimerEntities
- Methods
  - ClearProductionId(ReqClear clearData) : ResGeneral
  - EndTimer(ReqEnd endData) : ResGeneral
  - GetProductionOrder(long id) : ProductionOrder
  - PauseEnd(ReqEndPause endPauseData) : ResGeneral
  - PauseStart(ReqStartPause startPauseData) : ResGeneral
  - ProductionOrderRepository(ProdTimerEntities entities)
  - SeedData() : void
  - SetProductionId(ReqSet setData) : ResProductionInfo
  - StartTimer(ReqStart startData) : ResGeneral

ProdTimerEntities
Class
DbContext

- Properties
  - NoteBooks { get; set; } : DbSet<NoteBook>
  - Notes { get; set; } : DbSet<Note>
  - vProdOrdInfoes { get; set; } : DbSet<vProdOrdInfo>
  - WorkCells { get; set; } : DbSet<WorkCell>
- Methods
  - clearProductionData(long? productionId, string wSId) : ObjectResult<clearProductionData_Result>
  - endProductionTimer(long? productionId, DateTime? endTime, string workCell, int? numEmpl) : ObjectResult<endProductionTimer_Result>
  - getProductionData(long? productionId) : ObjectResult<vProdOrdInfo>
  - getProductionData(long? productionId, MergeOption mergeOption) : ObjectResult<vProdOrdInfo>
  - OnModelCreating(DbModelBuilder modelBuilder) : void
  - pauseEnd(long? productionId, DateTime? endTime, string workstationId) : ObjectResult<pauseEnd_Result>
  - pauseStart(long? productionId, DateTime? startTime, string reason, string workstationId) : ObjectResult<pauseStart_Result>
  - ProdTimerEntities()
  - seedProductionData() : int
  - setProductionOrder(long? productionId, string wSId) : ObjectResult<setProductionOrder_Result>
  - startProductionTimer(long? productionId, DateTime? startTime, string workCell, int? numEmpl) : ObjectResult<startProductionTimer_Result>

vProdOrdInfo
Class

- Properties
  - description { get; set; } : string
  - duration { get; set; } : int?
  - endTime { get; set; } : DateTime?
  - numberEmployees { get; set; } : int?
  - productionId { get; set; } : long
  - productNumber { get; set; } : string
  - quantity { get; set; } : int
  - startTime { get; set; } : DateTime?
  - status { get; set; } : string
  - statusId { get; set; } : int
  - workCell { get; set; } : string
  - workstationId { get; set; } : string

setProductionOrder_Result
Class

- Properties
  - Description { get; set; } : string
  - ProductionId { get; set; } : long?
  - ProductNumber { get; set; } : string
  - Quantity { get; set; } : int
  - ReturnCode { get; set; } : int?
  - ReturnMessage { get; set; } : string
  - Status { get; set; } : string
  - StatusId { get; set; } : int?

Note the rest of the Stored Procedure result classes are exactly like the startProductionTimer Result class.

startProductionTimer_Result
Class

- Properties
  - ReturnCode { get; set; } : int?
  - ReturnMessage { get; set; } : string

Production Order is a class that will be used to hold the results of getProductionData this will be used in future development.

ProductionOrder
Class

- Properties
  - Description { get; set; } : string
  - EndTime { get; set; } : DateTime?
  - Id { get; set; } : long
  - NumEmployee { get; set; } : int
  - Pauses { get; set; } : List<Paused>
  - ProductNumber { get; set; } : string
  - Quantity { get; set; } : int
  - StartTime { get; set; } : DateTime?
  - Status { get; set; } : ProductionOrderProcessStatus
  - WorkCell { get; set; } : string
  - WorkStationId { get; set; } : string
- Methods
  - ProductionOrder()

ProductionOrderProcessStatus
Enum

- unset
- Ready
- Set
- Started
- Completed

Paused
Class

- Properties
  - EndTime { get; set; } : DateTime
  - Reason { get; set; } : string
  - StartTime { get; set; } : DateTime
- Methods
  - Duration() : TimeSpan

Business and Entity Framework Classes

Business Model Classes

ProdTmrDataRepository

This class is used to allow for simple access to all of the data repositories available to the system. It is injected into the Production Timer Controller and provides a way to provide alternative data stores for tests or future development.

IProductionOrder and **IWorkCell** are interfaces that are implemented by the repository classes. These are defined so that other, alternate version repository classes, such as for a test project can be designed.

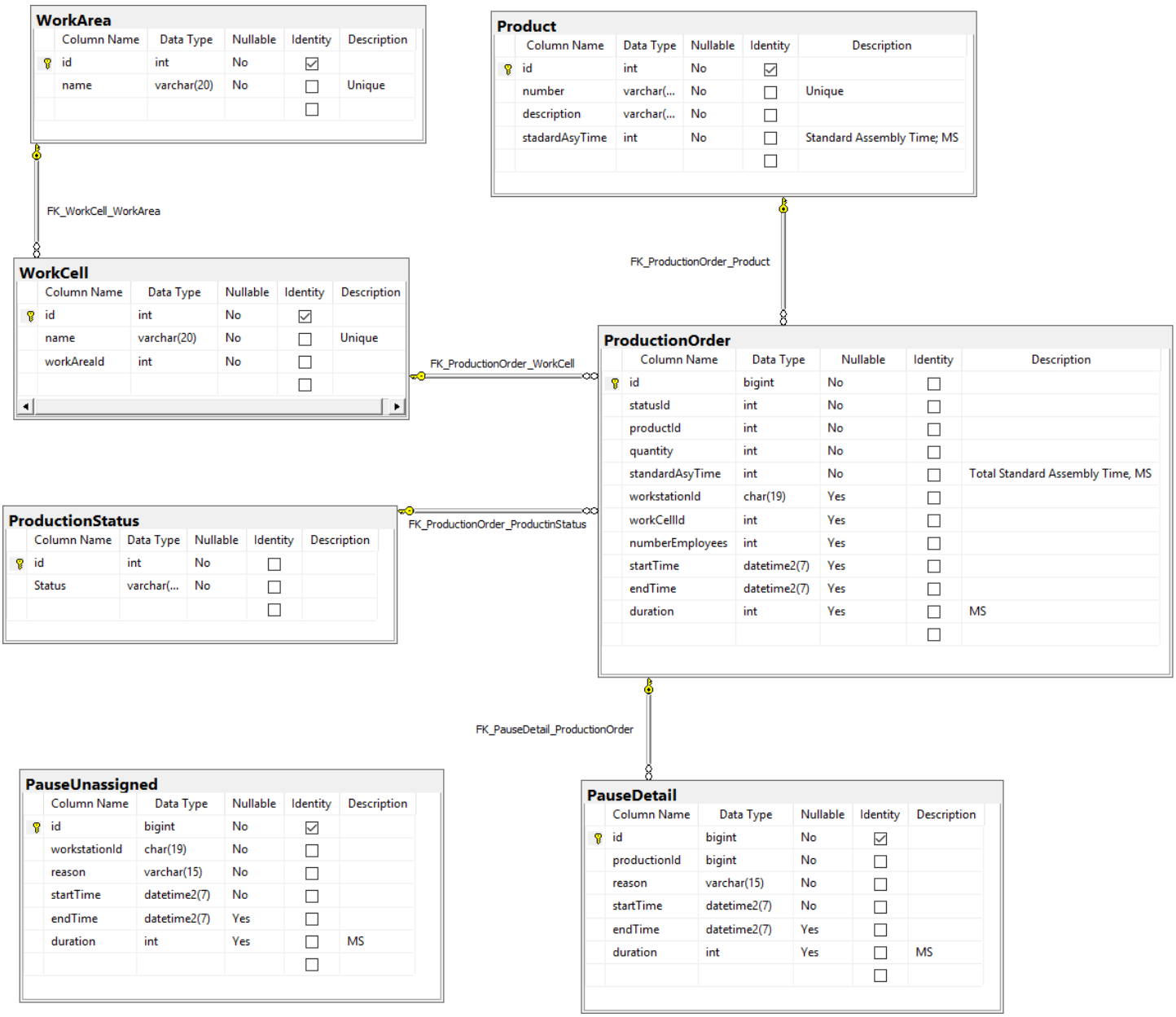
ProductionOrderRepository and **WorkCellRepository** are the primary classes that retrieve and process data. These classes makes calls to the Entity Framework class, which actually interfaces with the database.

Entity Framework Classes

The only class you deal with here is the ProdClassEntities class. This class is set up via the Entity Framework tools and should not be directly manipulated. The database is primarily communicated with via stored procedures. Most of the methods of this class require the same parameters that the stored procedure requires and returns the same results that the stored procedure returns.

Database Structure

Table Structure



Stored Procedures

The stored procedures are called by the ProdTimerEntities class methods. Entity Framework handles the connection to and communication with the data store, in this case a SQL Server database.

If you have questions or comments please contact me via email.