**Productivity Apex, Inc.**

Drayage Route Optimizer  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Projects Included In Solution**

|  |  |
| --- | --- |
| Core |  |
| PAI.Core | *- Common interfaces used throughout solution*  *Defines common elements that are composed of objects defined in either the System or System.Runtime.Serializaiton namespaces and a common IEntity interface which underpins a common means for identifying similar objects.* |
| PAI.FRATIS.Data | *- Data Access Tier: Entity Framework implementation*  *Defines three namespaces, PAI.FRATIS.Data, PAI.FRATIS.Data.Mappings, and PAI.FRATIS.Data.Migrations. PAI.FRATIS.Data comprises the EntityFramework implementation of the domain objects used by the Drayage Route Optimizer, the Mappings namespace defines any unique mappings necessary from the persistence repository objects to the domain objects. The Migrations namespace is used to create or modify seed data and contains method hooks for implementing run once data conversion methods.* |
| PAI.FRATIS.DateTimeService | *- Utility methods for UTC / Local time conversions*  *Defines a common interface to facilitate handling data of different or uncertain time zoning. Allows for easy time comparison and manipulation of time and date objects across time zones* |
| PAI.FRATIS.Domain | *- Business / Domain Objects*  *Comprises several subnamespaces: Congifuration, Equipment, Geography, Geography.NokiaMaps, Geography.NokiaMapos.TrafficItems, Information, Logging, Messaging, Orders, Planning, Times, and Users*  *The Configuration namespace contains configuration and preference objects; the Equipment namespace defines typically amortizable physical goods represented in the optimizer; Geography defines objects that represent either a physical location and situational effects that are geographically bound, such as traffic and weather; the NokiaMaps and NokiaMaps.TrafficItems namespaces further define traffic and location specific objects that are specific to users that are integrating with a Nokia Data Provider. Information defines those objects necessary for real time positioning; Logging provides facilities for outputting debugging and information messages to the file system or screen; Messaging provides an Alert object which can route a message or event from a user to another; Orders defines the objects that are necessary to instantiate an order to process through the optimization algorithm, such as drivers, jobs, jobs status updates, and which actions are available to undertake during a stop; Planning segregates thoseelements uses to set up and generate an optimization plan for a number of drivers with some number of jobs; Times supplies an access point for fine tuning actions by the day of th week; and Users defines the objects necessary for user management.* |
| PAI.FRATIS.Infrastructure | *- Project Infrastructure, Ninject Inversion of Control (IoC)*  *Infrastructure segregates the Infrastructure; Infrastructure.Data; Infrastructure.Engine; and Infrastructure.Threading objects that are used to implement the Dependency Injection pattern and support multi-threaded generation of the optimal route.* |
|  |  |
| Services |  |
| PAI.FRATIS.DataServices | *- Services for Domain Objects, Persistence*  *Provides an access point for application access of persistence repositories during execution of the optimization algorithm.* |
| PAI.FRATIS.DataServices.Core | *- Entity Services base implementations*  *Instantiates low level objects and services used by the PAI.FRATIS.DataServices.Core objects to manage cache and repository access.* |
|  |  |
| External Services |  |
| PAI.FRATIS.Services.ExternalDistanceService | *- Navteq implementation of pluggable external traffic / distance service*  *Provides an access point, plug-in framework for interfacing with external data sources.* |
| PAI.FRATIS.Wrappers.NokiaMaps | - Nokia Here / Maps external service implementation  An example implementation of the Nokia Mapping Service |
| PAI.FRATIS.Wrappers.QueueTimes | - Acyclica external service implementation for Terminal Queue reporting  An example implementation of an external Terminal Queue Reporting Service |
| PAI.FRATIS.Wrappers.WebFleet | - TomTom WEBFLEET external service implementation  An example implementation of the TomTom Mapping |
| PAI.FRATIS.Wrappers.YahooWeather | - Yahoo Weather external service implementation  An example implementation of the Yahoo Weather Service |
| PAI.FRATIS.YusenTerminalService | - Yusen Marine Terminal external service implementation  An example implementation of the Yusen Terminal Service |
|  |  |
| Optimization Algorithm |  |
| PAI.Drayage.Optimization | - Optimization Algorithm  The optimization algorithm, wherein jobs and drivers are matched on user provided selection criteria. Typically this is minimizes either time or distance traveled while servicing as many jobs as possible. |
| PAI.Optimization.Adapter | - “Adapter” intermediary between domain objects and algorithm  Access point for further configuration and adjustment of the optimization algorithm or supporting object after initialization and during run-time. |
| PAI.Drayage.Optimization.Reporting | - Reporting and statistical service for optimization algorithm solutions  A collection of objects and methods that will examine generated routes and schedules to extract meaningful data over the generated route solutions. |
|  |  |
| Executable Processes |  |
| PAI.FRATIS.BackgroundProcesses | - Background process wrappers for container availability, traffic/travel time  Supplimental process that run in worker threads supporting the creation and maintenance of relatively static and parallelizable data sources and elements |
| PAI.FRATIS.ConsoleDemo | - Demo application  A simple console application that demonstrates how to provide data to and consume the results of the optimization algorithm. Uses a random subset of a static data set to create drivers and orders to feed into the algorithm. Additionally illustrates how to consume the results of the optimizer. |
|  |  |
| Tests |  |
| PAI.Drayage.Tests | Test base classes and Extension methods |
| PAI.FRATIS.ExternalServices.Tests | Integration tests for external data source connections |
| PAI.FRATIS.Tests | Unit tests to ensure the correct behavior of the component objects of the project whole. |

**Summary**

The Drayage Optimizer will produce a solution set that represents the best determined sequence of stops based upon **Locations**, **Drivers**, **Jobs** and **Routes** provided to the algorithm.

The solution is built leveraging v4.5 of the .NET Framework. The optimization class library has a small footprint, and should run successfully on any computer capable of supporting .NET v4.5.

**Inline Code Comments**

Class and property summaries, as well as incline code comments are provided for key logic within this solution. Accompanying unit and integration tests, as well as a sample executable application (PAI.FRATIS.ConsoleDemo) are also provided to demonstrate usage.

**Drayage Optimizer Dependencies**

This optimization algorithm, itself, has no external dependencies aside from the utilization of a Dependency Injector (ex: Ninject, Unity, Castle Windsor) to facilitate Inversion of Control. This process has been demonstrated in the included sample executable with open-source solution “Ninject”.

The additional layers can be implemented on top of the optimization algorithm to readily allow for object persistence, traffic and driving condition analysis via external services, and marine terminal queue time querying.

**Drayage Optimizer Objects**

Within the PAI.Drayage.Optimization project, the Model namespace contains all of the objects that are required in order to build a solution. These key objects will be explained later in this document. At a very high level, the tool requires **Drivers**, **Jobs**, **RouteStops**, and **Locations** in order to produce an optimized solution.

**Database Schema**

The Optimization Algorithm does not require a database in order to properly build solutions. As demonstrated in the sample executable, object can be manually initialized in order to produce a solution without the dependency of a database. If desired, object persistence can easily be achieved using any .NET supported database type and ORM, based upon project requirements and desired implementations.

Within the PAI.FRATIS.Data project of this solution is an implementation of the Microsoft Entity Framework repository pattern. This can be used to persist domain objects to a database, which can later be queried and mapped to the optimization algorithm to produce a solution from saved data.

**Overview of Drayage Optimizer Key Business Objects**

**(PAI.Drayage.Optimization.Model.Location)  
Location:** a representation of latitude and longitude points that relate to a given location. The Location object is used to represent the address of each of the route stops. The location is identified by the DisplayName property, and the Latitude and Longitude properties are used to represent the coordinates. Each Driver is also assigned a StartingLocation.

**(PAI.Drayage.Optimization.Model.Orders.Driver)  
Driver:** represents a driver that is able to complete a Job. Essential properties are DisplayName (name/nickname), AvailableDrivingHours (maximum TimeSpan for driving in a given day), AvailableDutyHours (maximum TimeSpan for allowable work hours within a day), EarliestStartTime (TimeSpan representing the earliest a Driver can work), and StartingLocation (representing the driver’s starting point).

**(PAI.Drayage.Optimization.Model.Orders.Job)  
Job:** represents a series of load / unloads to be performed by a Driver to satisfy the delivery requirements for a client order. DisplayName identifies the Job, while RouteStops is a collection representing each “stop” of the job (minimum of 2 RouteStops must be provided for each job [pickup and drop-off]).

**(PAI.Drayage.Optimization.Model.Orders.RouteStop)  
RouteStop:** represents each stop / leg of a Job. Complemented by a StopAction which determines the specific action to be performed at each stop. RouteStop objects also have properties for Location (the coordinates of the load/unload), StopDelay (optional – representing the maximum anticipated duration of the stop while on-location), WindowStart and WindowEnd (the earliest and latest time window that a load/unload can be performed at this location).

**(PAI.Drayage.Optimization.Model.Orders.StopAction)  
StopAction:** represents the load/unload action that is to be performed at each stop. Available actions are: PickupChassis, DropOffChassis, PickupEmpty, DropOffEmpty, PickupEmptyWithChsassis, DropOffEmptyWithChassis, PickupLoaded, DropOffLoaded, PickupLoadedWithChassis, DropOffLoadedWithChassis, LiveLoading, LiveUnloading. The sequence of these actions must logically be valid in order for the Job to be processed (ex: if RouteStop 1’s action is PickUpLoadedWithChasis, RouteStop 2 cannot be DropOffEmpty, as the truck is already loaded and is not in the Empty state).  
 **LocationDistance:** represents the actual travel distance and travel time between two locations based upon current conditions reported by an external mapping / traffic provider. This record can store travel times for every hour of the day within the **Hours** property. These travel times can be used by the algorithm to provide the most accurate driving estimations for all hours of a given day.

**LocationQueueDelay:** represents a delay at a given location during a specified window of time. This is particularly useful for allowing the algorithm to consider Marine Terminal queue delays when entering a facility. These days are assumed to be assessed after arriving at the location/destination, but before the execution of the given job (ex: waiting at an entrance gate).

**Domain Objects**

The **PAI.FRATIS.Domain** project contains the domain / business objects that may be used for object persistence. A corresponding **DataService** exists for each of these domain objects in the **PAI.FRATIS.DataService** project. All data service classes extend PAI.FRATIS.DataServices.Core.**EntityServiceBase** to provide a unified way to query, manipulate and save business objects. Inline summaries and comments of key properties are provided.

**Optimization Algorithm Adapter**

Since domain objects cannot be provided to the optimization algorithm directly (rather, PAI.Drayage.Optimizer.Model objects must be instantiated), an intermediary project / adapter solution was created in order to facilitate the passing of domain objects to the Drayage Optimizer. Using the **PlanGenerator** service within PAI.Drayage.Optimizer.Adapter project, domain objects can readily be passed to the optimizer. The **PlanGenerator** has methods that accept both PAI.FRATIS.Domain objects, as well as PAI.Drayage.Optimizer.Model objects – so this can be the entry point for consuming the algorithm for either use case. The domain objects will be mapped to the optimization model using the Omu.ValueInjecter **MapperService**.

**Background Processes**

Background processes have been created to demonstrate updating travel times, traffic conditions, and marine terminal container availability in the background.

* PAI.FRATIS.BackgroundProcesses.**ContainerAvailabilityBW**
  + Queries the Marine Terminal Service for domain jobs that are identified as Unavailable / Unassigned, and updates their status if their availability is confirmed.
  + Updating the **Job** status to IsReady=true identifies that the marine terminal pickup/dropoff contained within this job is now ready.
* PAI.FRATIS.BackgroundProcesses.**LocationDistanceCreater**
  + When a new order is persisted as a domain **Job** with a due date, this background worker identifies the potential locations that may also be executed on that same day. Accordingly, this process creates empty **LocationDistance** records that contain all of the origin and destination points. These empty records will be used by the **LocationDistanceUpdater** process to populate them with relevant travel conditions that can be used by the algorithm.
* PAI.FRATIS.BackgroundProcesses.**LocationDistanceUpdater**
  + This process looks for empty **LocationDistance** records, and queries the **ExternalDistanceService** to determine the actual travel times, considering traffic, for various times of the day. The domain object is updated and saved, allowing for utilization by the algorithm.

**Console Demo**

A demonstration console application has been included in this solution to demonstrate the instantiation of core objects, and how to build an optimized solution using the **PlanGenerator**. This process will be demonstrated in the sample code excerpts below.

**Sample Code – DrayageOptimizer.cs Sample Usage**

Step 1: Initialize the Locations

// STEP 1 - Define the locations to be used by the Optimizer

// This includes all starting points, end points,

// route stop locations, driver starting locations

// driver starting location

var startLocation = new Location

{

DisplayName = "Home",

Latitude = 35.006,

Longitude = -89.8913

};

// route stop locations

var location1 = new Location

{

DisplayName = "Location 1",

Latitude = 35.0803,

Longitude = -89.963

};

Step 2: Initialize the Drivers

var d1 = new Driver

{

AvailableDrivingHours = 11.0,

AvailableDutyHours = 14.0,

DisplayName = "Driver 1",

EarliestStartTime = Helper.GetTimeSpan(6, 0),

StartingLocation = startLocation

};

Step 3: Create Job(s)

var job = new Job

{

DisplayName = "Job 1",

Id = 1

};

Step 4: Add Route Stops to the Job with corresponding StopActions.

job.RouteStops = new List<RouteStop>

{

Helper.CreateRouteStop(job, StopActions.PickupEmptyWithChassis, location1,

60, Helper.GetTimeSpan(6, 0), Helper.GetTimeSpan(22, 0)),

Helper.CreateRouteStop(job, StopActions.DropOffEmpty, location2,

60, Helper.GetTimeSpan(6, 0), Helper.GetTimeSpan(22, 0)),

Helper.CreateRouteStop(job, StopActions.PickupLoadedWithChassis, location2,

90, Helper.GetTimeSpan(6, 0), Helper.GetTimeSpan(22, 0)),

Helper.CreateRouteStop(job, StopActions.DropOffLoadedWithChassis, location3,

60, Helper.GetTimeSpan(6, 0), Helper.GetTimeSpan(22, 0))

};

Step 5: Place all created Jobs and Drivers within a collection.

var jobs = new List<Job>() {job, job2};

var drivers = new List<Driver>() {d1, d2};

Step 6: Initialize the Optimizer and generate the solution.

var optimizer = GetService<IDrayageOptimizer>();

optimizer.Initialize();

// build the solution

var solution = optimizer.BuildSolution(drivers, drivers[0], jobs);

Step 7: Output solution results:

Console.WriteLine("Solution Created");

Console.WriteLine(solution.RouteSolutions.Count + " route solutions.");

Console.WriteLine(solution.UnassignedJobNodes.Count + " unassigned jobs.");

Console.WriteLine(solution.RouteStatistics.TotalTime + " : total time.");

int count = 0;

foreach (var routeSolution in solution.RouteSolutions)

{

count++;

Console.WriteLine("\n\tROUTE #" + count);

Console.WriteLine("\n\tAssigned to Driver: " + routeSolution.DriverNode.Driver.DisplayName);

Console.WriteLine("\t\tTravel Distance: " + routeSolution.RouteStatistics.TotalTravelDistance + " miles");

Console.WriteLine("\t\tEstimated Total Time: " + routeSolution.RouteStatistics.TotalTime);

Console.WriteLine("\t\tEstimated Travel Time: " + routeSolution.RouteStatistics.TotalTravelTime);

Console.WriteLine("\t\tNodes:");

foreach (var node in routeSolution.AllNodes)

{

foreach (var routeStop in node.RouteStops)

{

Console.Write("\t\t\t" + routeStop.Location.DisplayName);

if (routeStop.StopDelay.HasValue)

Console.Write(" - " + routeStop.StopDelay.Value.TotalMinutes + " minute stop" );

Console.Write("\n");

}

}

}

**Sample Code – Drayage Optimizer Adapter**

- Use the **MockData** class to create sample Locations, Drivers and Jobs (which respectively contain RouteStops).

var startLocation = MockData.GetMockLocations(1, "Driver").FirstOrDefault();

var locations = MockData.GetMockLocations(6);

var drivers = MockData.GetMockDrivers(3, startLocation);

var jobs = MockData.GetJobs(8, locations, true);

- Initialize the PAI.Drayage.Optimizer.Adapter.**PlanGenerator** service.

var planGenerator = GetService<IPlanGenerator>();

- Build an optimized solution.

var solution = planGenerator.GeneratePlan(drivers, jobs, DayOfWeek.Saturday, queueDelays);

- Initialize the PAI.Drayage.Optimizer.Reporting.**ReportingService** and get statistics.

var reportingService = GetService<IReportingService>();

var solutionPerformance = reportingService.GetSolutionPerformanceStatistics(solution);

var driverStatistics = solutionPerformance.TruckStatistics.FirstOrDefault(

p => p.Key.DriverNode.Driver.Id == driverId).Value;

**Sample Code – Travel Time with Traffic**

The **IExternalDistanceService** interface can be implemented with any mapping / traffic service. The out-of-box implementation is with Navteq Maps / Nokia Here. In the following example, we will obtain the actual traffic distance between two points. The returned result from the external service will contain traffic conditions at various times of the day, which can be accessed using the Hours[] array.

Hours[0] represents UTC 12:00 AM  
Hours[23] represents UTC 23:00 PM

# LocationDistance Get(Location startLocation, Location endLocation, DateTime departureDay, LocationDistance record = null);