# SDD

# Wayne.ForecourtControl

# Table of Contents

1 Document information 4
1.1 Revision history 4
1.2 Purpose and scope 4
1.3 Abbreviations and acronyms 4
1.4 References 4
1.5 COM interop 4
2 Overview 5
3 Forecourt Control 6
4 Pumps 7
4.1 Pump states 8
5 Fuellings 9
5.1 Fuelling states 10
5.2 Safe handling of the Fuelling objects 11
5.2.1 AuthorizationId 11
5.2.2 Current Fuelling 11
5.2.3 Fuellings array 11
5.2.4 Events 11
5.2.5 Housekeeping 11
5.2.6 Integrity of the fuelling data 11
6 Price handling 12
6.1.1 Changing the prices 12
6.1.2 Fuel periods 12
7 Tanks 13
8 Configuration 14
9 Namespace Wayne.ForecourtControl 15
9.1 Interfaces 16
9.1.1 Interface IAllowedFuelGrades 17
9.1.2 Interface IAuthorizeParameters 17
9.1.3 Interface IForecourtConfig 17
9.1.4 Interface IForecourtControl 18
9.1.5 Interface IFuelling 20
9.1.6 Interface IFuelPrice 22
9.1.7 Interface IFuelPriceAddPricePerPriceGroup 22
9.1.8 Interface IManualFuelDeliveryParameters 22
9.1.9 Interface INozzle 23
9.1.10 Interface IPricePole 23
9.1.11 Interface IPricePoleSegment 24
9.1.12 Interface IPump 24
9.1.13 Interface ITank 28
9.1.14 Interface ITankGroup 29
9.1.15 Interface ITankReading 30
9.2 Classes 31
9.2.1 Class AlarmEventArgs 31
9.2.2 Class AllowedFuelGrades 32
9.2.3 Class AuthorizeParameters 32
9.2.4 Class ForecourtControlException 32
9.2.5 Class ForecourtControlXml 33
9.2.6 Class FuelDeliveryEventArgs 33
9.2.7 Class FuelGradeOutOfRangeException 34
9.2.8 Class FuellingData 35
9.2.9 Class FuellingDataChangeEventArgs 36
9.2.10 Class FuellingStateChangeEventArgs 36
9.2.11 Class ManualFuelDeliveryParameters 37
9.2.12 Class NozzleStateChangeEventArgs 37
9.2.13 Class PumpAccumulatorReading 38
9.2.14 Class PumpEventOccuredEventArgs 39

9.2.15 Class PumpStateChangeEventArgs 39
9.2.16 Class SiteModeChangeEventArgs 40
9.3 Enumerations 40
9.3.1 Enumeration FuelDeliveryType 40
9.3.2 Enumeration FuellingState 40
9.3.3 Enumeration FuellingType 41
9.3.4 Enumeration NozzleState 41
9.3.5 Enumeration PresetType 41
9.3.6 Enumeration ProbeState 41
9.3.7 Enumeration PumpAccumulatorReadingType 42
9.3.8 Enumeration PumpEventType 42
9.3.9 Enumeration PumpState 42
9.3.10 Enumeration UnitOfMeasure 42
10 Namespace Wayne.ForecourtControl.Com 43
10.1 Interfaces 44
10.1.1 Interface IAlarmEventArgs 44
10.1.2 Interface IForecourtControl 44
10.1.3 Interface IForecourtControlEvents 46
10.1.4 Interface IFuelDeliveryEventArgs 47
10.1.5 Interface IFuelling 48
10.1.6 Interface IFuellingEvents 50
10.1.7 Interface INozzle 51
10.1.8 Interface INozzleEvents 51
10.1.9 Interface IPricePole 52
10.1.10 Interface IPricePoleEvents 52
10.1.11 Interface IPricePoleSegment 52
10.1.12 Interface IPump 53
10.1.13 Interface IPumpAccumulatorReading 56
10.1.14 Interface IPumpEvents 56
10.1.15 Interface ITank 59
10.1.16 Interface ITankEvents 60
10.1.17 Interface ITankGroup 61
10.1.18 Interface ITankGroupEvents 61
10.2 Enumerations 62
10.2.1 Enumeration DeviceConnectionState 62

# 1 Document information

File: SDD_Wayne.ForecourtControl.doc

## 1.1 Revision history

| Revision | Author | Date | Change description |
|---|---|---|---|
| 2.0 | Roger Månsson | 2006-07-25 | Created |
| 2.1 | Roger Månsson | 2007-01-09 | Added COM interfaces, tanks and some more descriptions. |
| 2.2 | Roger Månsson | 2007-03-09 | Updated with some PricePole and TankGroup things. |
| 2.3 | Roger Månsson | 2011-06-13 | Added section about securely handle Fuelling objects. |

## 1.2 Purpose and scope

## 1.3 Abbreviations and acronyms

| Abbreviation | Meaning |
|---|---|
| | |
| | |

## 1.4 References

## 1.5 COM interop

The Forecourt control library is COM – compatible, but the COM interop needs to have its separate classes. These are put under the namespace Wayne.ForecourtControl.Com. The types that can be used in both .Net and COM are in the normal Wayne.ForecourtControl namespace. This document describes mainly the .Net interface, although the COM interfaces functionality is the same although the syntax is a bit different. The COM interfaces are specified in chapter 10 of this document.

For unsolicited events, i.e. normal .Net events like OnConnectionStateChange are translated into methods in a separate interface. The interface is suffixed ….Events. See IForecourtControl and IForecourtControlEvents. The event sink in the COM Client then should implement this interface.

All methods in the interfaces are asynchronous, and the Wayne standard is to have a signature

```
void DoSomethingAsync(int aParameter,
                      EventHandler<AsyncCompletedEventArgs> e,
                      object userToken);
```

When the asynchronous method completes, the supplied EventHandler delegate (callback) is invoked. In that invocation, an object of the type AsyncCompletedEventArgs will be supplied. It contains an indicator on if the request was successful, and will also return the user token that was supplied in the request.

In COM it is not allowed to have delegates (callback pointers) as parameters to methods, so the signature for the method will be:

```
void DoSomethingAsync(int aParameter, object userToken)
```

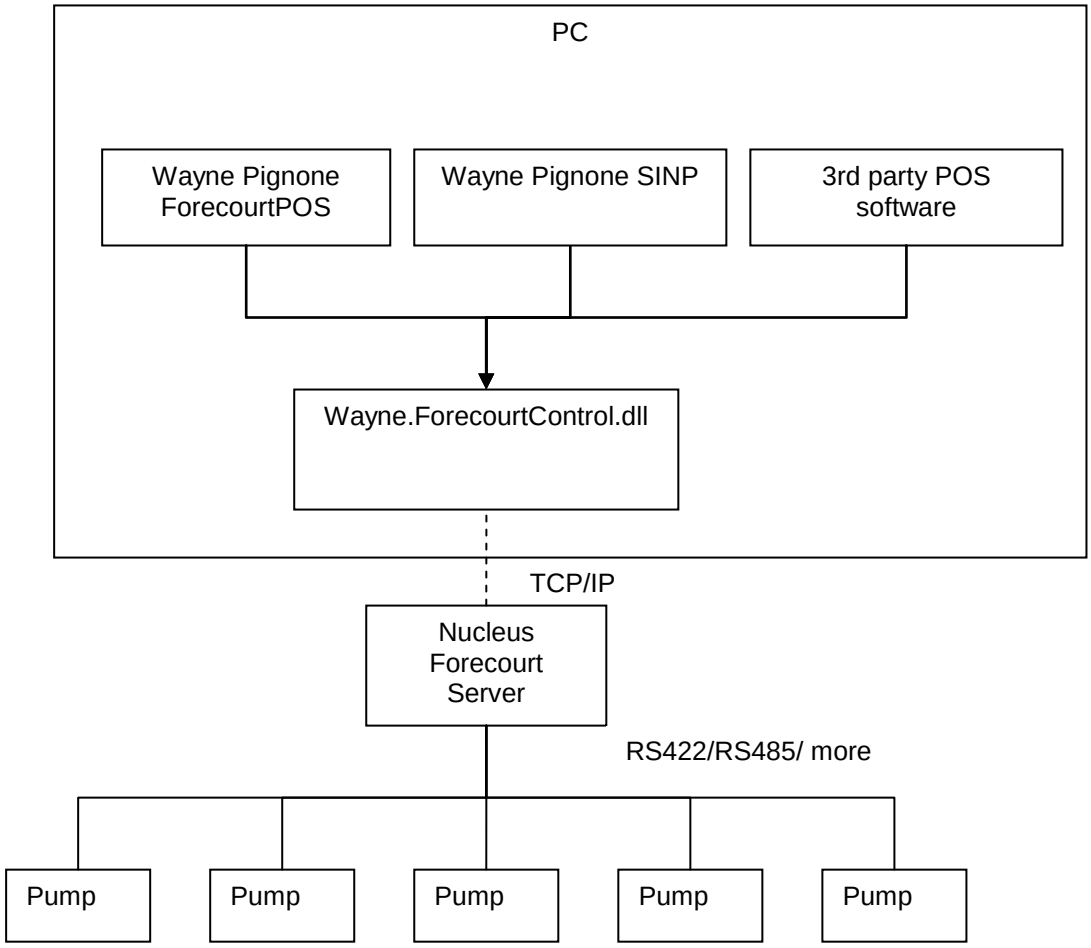
To be able to notify when the asynchronous method completes, we define a corresponding event in the event interface:

```
void OnDoSomethingCompleted(object sender, bool success, object userToken)
```

This will be invoked when the asynchronous request completes.

# 2 Overview

The forecourt control library enables POS software to manage pumps and fuelling on a high level. It lets the POS software work with a high level interface that hides the complexity in the variety of brands and models of dispensers and different protocols.



# 3 Forecourt Control

The forecourt control object is the main object in the library. It contains some station-wide properties like site mode and if the site is open. It does also have functionality for manipulating the fuel prices used in the forecourt. It does also contain the collections of pumps and tank groups.

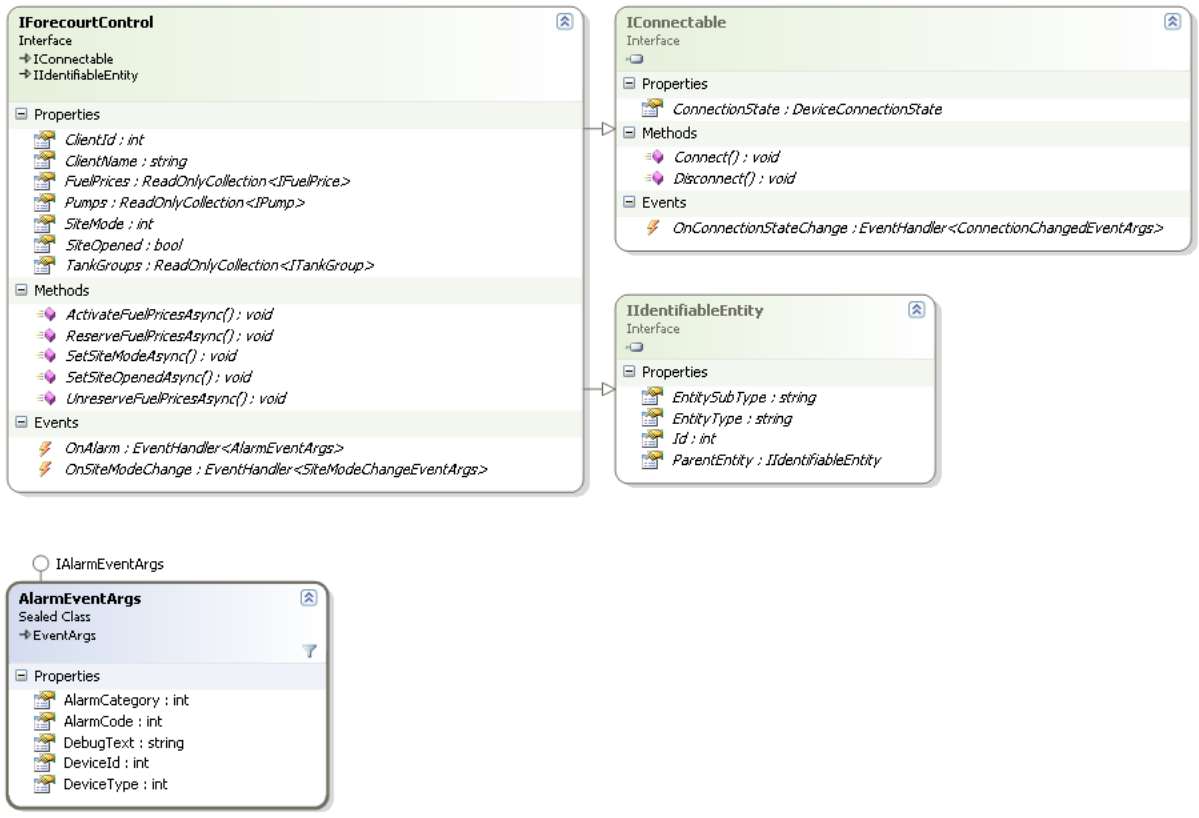
After creating the forecourt control object, it is still not initialized, and has the ConnectionState=*Disconnected*. In order to perform any operations, the forecourt control must be connected to the forecourt server. This is performed using the *Connect*() method using a connection string. When connecting to the forecourt server, the connection string is a comma-separated string with the connection parameters.

**Example:**

```
ClientId=210,ClientName=TestApp,Host=192.168.1.1
```

| Parameter | Range | Description |
|---|---|---|
| ClientId | 1-65,535 | A unique client number.<br>0 is not allowed, and 200-205 is reserved for internal use in the Nucleus application. |
| Client name | --- | A string representing the application name. This is only used for debug purposes and can be left out. |
| Host | | IP Address or DNS name of the forecourt server to connect to. |

When connect has been called, the connection state will change to *Connecting*. When the forecourt control is finally connected and completely initialized, the connection state will change again to *Connected.* Connection state changes are notified using the OnConnectionStateChanged event.



# 4 Pumps

Most of the handling is done using the pump object. When the forecourt control is connected, the pumps that are configured to be connected are accessible in the *IForecourtControl.Pumps* array.

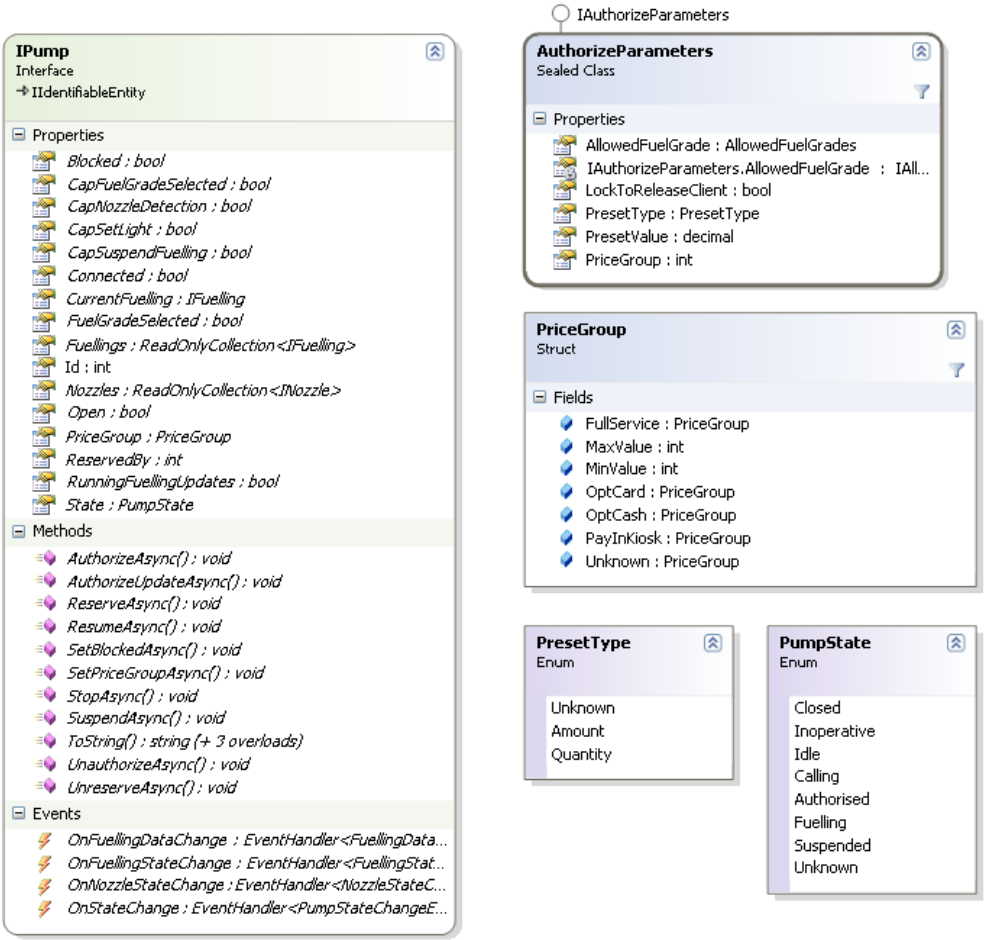
The pump object shows the current state of the pump, and has references to a number of *IFuelling* objects. There is one property in the IPump interface, CurrentFuelling that represents what happens currently on the pump. If the pump is idle, the fuelling will not contain any valuable data. If the pump is fuelling, the current fuelling object will contain some information about the running fuelling. If the property IPump.RunningFuellingUpdates is set to true, the volume and amount will be updated continuously during the fuelling for the current fuelling. After the fuelling is finished and until the pump is reset, the current fuelling will contain the latest pump data.

If we want to fuel on the pump, the pump must first be reserved. When reserving the pump, the intended fuelling type should be supplied. If the reservation succeeds, we are able to authorize the pump for a fuelling. That is done by calling the *AuthorizeAsync* method. In this method we supply an AuthorizeParams object. In this we specify

- Preset type (Volume or amount),
- Preset value,
- Which fuel grades to allow,
- Price group the fuelling should use,
- Flag to indicate if the fuelling can be paid by another client after the fuelling is finished.

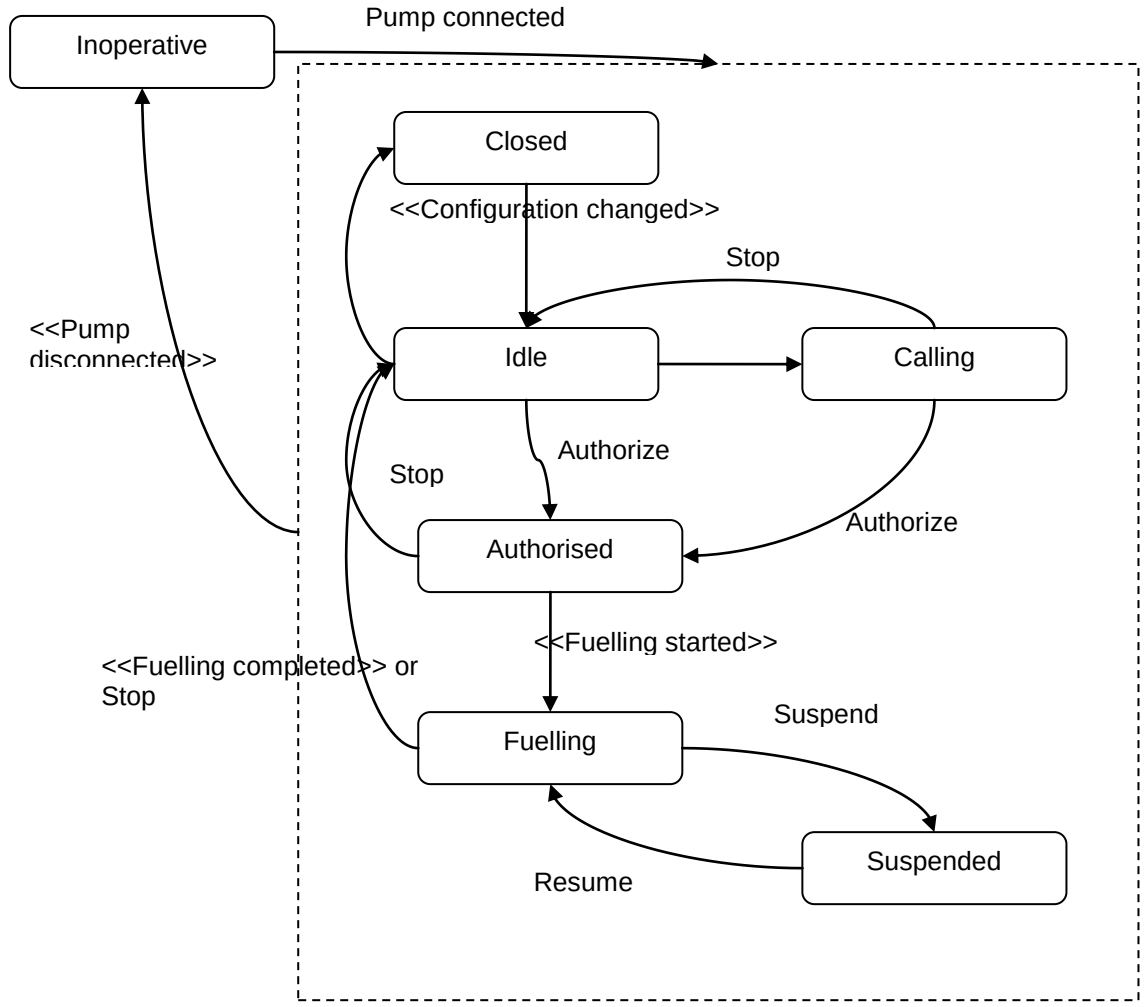
After an authorization the pump is automatically unreserved from the client so that is not necessary to do explicitly.

When the fuelling has finished, a new fuelling object is added to the Fuellings array. This should be removed, i.e. paid, by any of the clients. That is described in the Fuellings section of this document.



## 4.1 Pump states

This state schema shows the pump states.

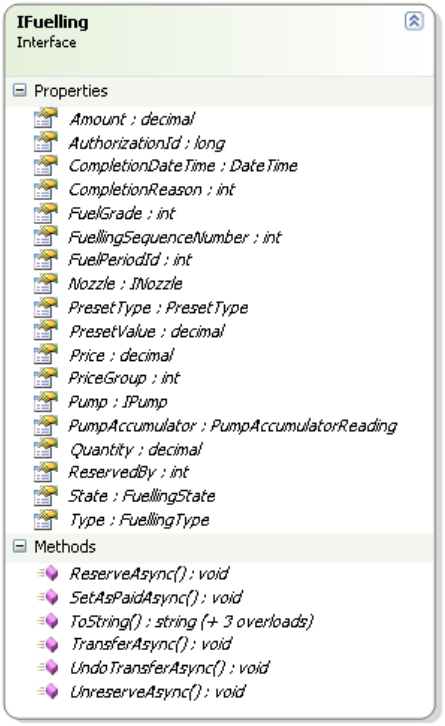


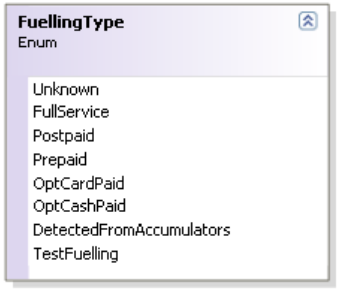
# 5 Fuellings

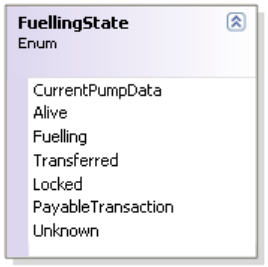
A fuelling object represents one fuelling and contains information about that as filled volume and amount. It does also provide the methods to set the fuelling as paid in a controlled procedure. The pump has fuellings accessible in three ways. The *CurrentFuelling* property contains one fuelling that represents the current pump data. It can be in three states; *Currentpumpdata* when it is in idle, *Alive* when the pump is reserved or authorized for a fuelling and *Fuelling* when the fuelling is running. When the fuelling is completed, the fuelling is added to the Fuellings array. In order to remove the fuelling from that array, the application must at the end call *SetAsPaid.* It can go by PayableTransaction and Locked if that is in any value to the application.

The normal procedure when removing a fuelling from a pump is to

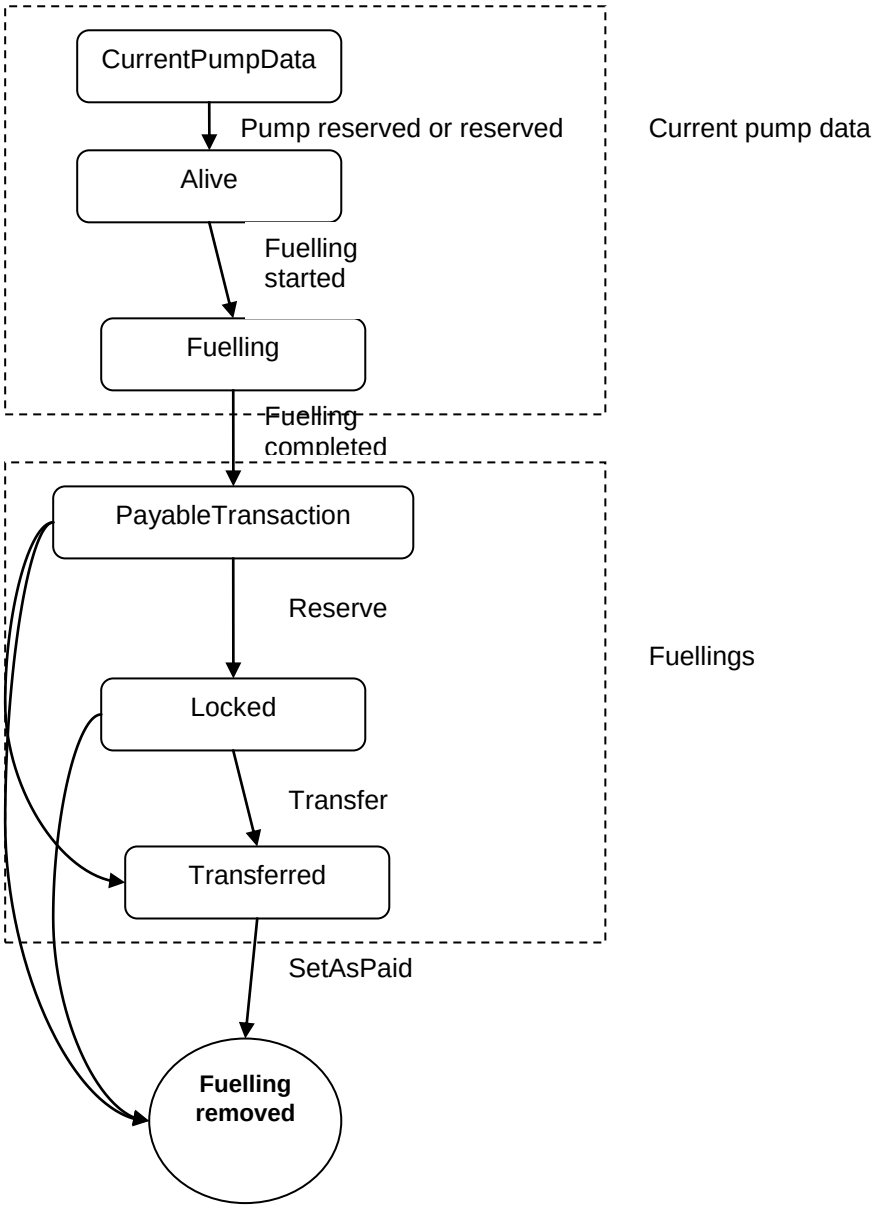
1. Reserve it. Exclusively reserve it so no other client can set it as paid at the same time.
2. Transfer it. Mark the fuelling that it is transferred to a payment handler like a Point of Sale.
3. Set as paid (Remove the fuelling). The fuelling is now paid and is removed and moved to historical fuellings array.



## 5.1 Fuelling states



## 5.2 Safe handling of the Fuelling objects

The Fuelling objects are important for billing of the transaction, either it is on a POS or an outdoor payment terminal. This section describes the preferred method of working with the Fuelling objects.

### 5.2.1 AuthorizationId

When authorizing the pump for a fuelling, the response event contains an AuthorizationId. This number should be stored persistently until the fuelling is completed. Then it has to be **matched with the fuelling** when the fuelling is complete. The Forecourt Controller guarantees that there will be one fuelling for each successful authorization. There are two exceptions

- In some exceptional cases no fuelling gets generated. The application code must have a reasonable timeout for waiting for a fuelling, N9 uses 12 hours. Note that under some circumstances it can take a long time for the fuelling to appear and therefore the timeout should not be too short as we would then bill the transaction with 0 unnecessarily.
- If the forecourt controller gets restarted during refueling, authorization id might be set to 0. This case must be handled, either manually or through automatic decision making. N9 decides based on pump number and date and time if we have one authorizationid 0 and exactly one transaction waiting for fuelling data on one pump. Any other case must be resolved manually.

An application must have logic to handle both the general case with authorization id, and the exceptional cases.

### 5.2.2 Current Fuelling

During a fuelling the progress of the fuelling can be checked on the IFuelling.CurrentFuelling object. However, this **must never be used to make any kind of business decisions**. Only fuellings found in the IPump.Fuellings array may be used for billing.

### 5.2.3 Fuellings array

The IPump.Fuellings is the list of unpaid transactions. If the site is idle, this array should be empty.

Note that it is only fuelling objects that are in the Fuelling array that can be used for billing.

### 5.2.4 Events

The events that get generated from the pump objects should be **used as triggers**. The preferred method of handling the fuel transactions is to work against the Fuellings array.

### 5.2.5 Housekeeping

The application should have a housekeeping task that checks for fuellings that are released by the applications (check Type and ReservingDeviceId). If fuellings are found that are not recognized, they should be removed and billed manually.

### 5.2.6 Integrity of the fuelling data

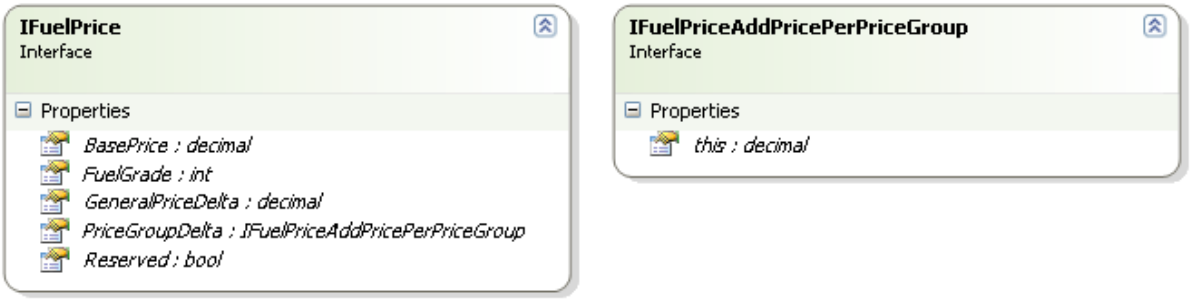
Due to Weight-and measurement reasons it is forbidden to alter the data that is in the fuelling object, even if the amount is calculated wrong. It should be printed on the receipt exactly as it is seen on the pump display.

# 6 Price handling

The price setup is common to the entire forecourt. The actual price is calculated with two parameters: The fuel grade and the price group. From the fuel grade, the base price is added with the general price change. Then, depending on the selected price group, the addprice for that price group is added to or withdrawn from the price.

| **FuelGrade** | Base Price | General price change | AddPrice Pricegroup 1 (FullService) | AddPrice Price group 2 (PayInKiosk) | AddPrice Price group 3 (OPT Card) | … |
|---|---|---|---|---|---|---|
| 1 | 1,10 | +0,2 | -0,01 | +0,02 | +0,03 | |
| 2 | 1,20 | -0,1 | -0,01 | +0,02 | +0,03 | |
| 3 | 1,03 | +0,4 | -0,01 | +0,02 | +0,03 | |
| 4 | 0,99 | -0,5 | -0,01 | +0,02 | +0,03 | |
| 5 | 0,40 | 0,10 | -0,01 | +0,02 | +0,03 | |
| …. | | | | | | |

ActualPrice = BasePrice + General Price Change + AddPrice[PriceGroup].




### 6.1.1 Changing the prices

The first thing is to Reserve the exclusive right to change the prices through the IForecourtControl.ReserveFuelPricesAsync method.

If it succeeds, the fuel price objects in the fuel price collection can be changed. They have properties for each parameter. Each Fuel price object corresponds to one fuel grade.

If we want to cancel the changes and leave the price setup as it was, we call IForecourtControl.UnreserveFuelPricesAsync

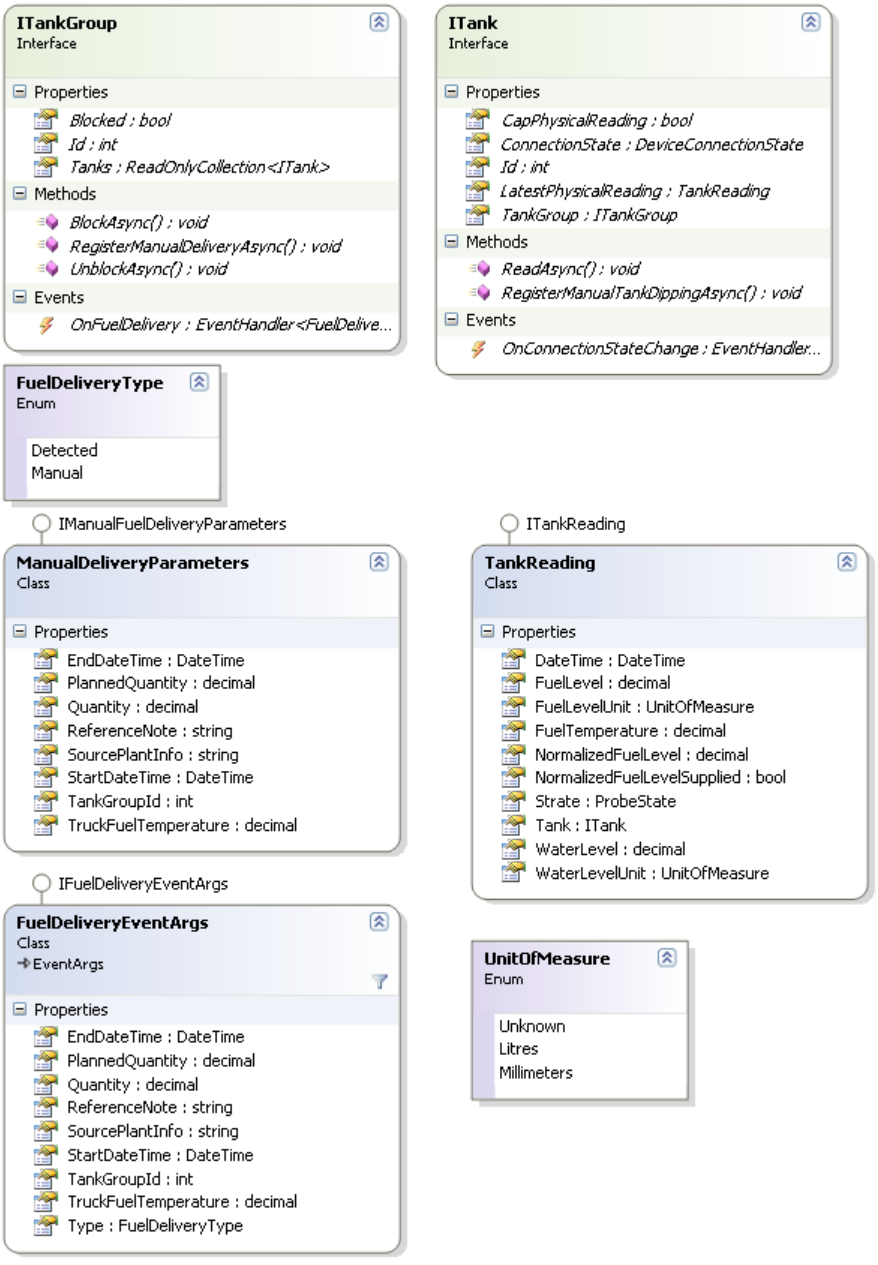
If we want to confirm all the changes made we call IForecourtControl.ActivateFuelPrices. This method will implicitly unreserve the fuel prices and activate the prices on the pump. It can take some time before the prices actually reaches the pump.

### 6.1.2 Fuel periods

One fuel period runs between two price changes. Each fuelling is marked with the fuel period it belongs to. When the prices are activated, the current fuel period is closed and a new is opened.

# 7 Tanks

The tank interfaces are used to manage the fuel stock, and to keep track of the connection state of the Tank probe equipment. The tanks are always linked to a tank group. A tank group contains one or more physical fuel tank. Fuel deliveries are made to a tank group, but tank readings are made in each tank.



# 8 Configuration

Configuring the site is performed through an XML script. The XML Schema ForecourtConfig.xsd defines the structure. The interface IForecourtConfig has two methods, ReadConfiguration and WriteConfiguration. These enables the application to change the setup such as fuel grades, pumps, nozzles etc.

Here is an example of the beginning of the file.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Forecourt Partial="false" xmlns="http://www.wayne.se/2006-03-
13/ForecourtConfig.xsd">
  <Pumps>
    <Pump Id="0">
      <Used>true</Used>
      <Nozzles>
        <Nozzle Id="0">
          <Used>true</Used>
          <FuelGradeId>0</FuelGradeId>
        </Nozzle>
        <Nozzle Id="1">
          <Used>true</Used>
          <FuelGradeId>1</FuelGradeId>
          <PrimaryTankId>0</PrimaryTankId>
        </Nozzle>
        <Nozzle Id="2">
          <Used>true</Used>
          <FuelGradeId>2</FuelGradeId>
          <PrimaryTankId>1</PrimaryTankId>
        </Nozzle>
        <Nozzle Id="3">
          <Used>false</Used>
        </Nozzle>
        <Nozzle Id="4">
          <Used>false</Used>
        </Nozzle>
        <Nozzle Id="5">
          <Used>false</Used>
        </Nozzle>
        <Nozzle Id="6">
          <Used>false</Used>
        </Nozzle>
        <Nozzle Id="7">
          <Used>false</Used>
        </Nozzle>
        <Nozzle Id="8">
          <Used>false</Used>
        </Nozzle>
        <Nozzle Id="9">
          <Used>false</Used>
        </Nozzle>
      </Nozzles>
    </Pump>
    <Pump Id="1">
      <Used>false</Used>
      <Nozzles>
        <Nozzle Id="0">
          <Used>true</Used>
          <FuelGrade>0</FuelGrade>
          <PrimaryTankId>0</PrimaryTankId>
        </Nozzle>
        <Nozzle Id="1">
```

# 9 Namespace Wayne.ForecourtControl

## Interfaces

| | |
|---|---|
| **IAllowedFuelGrades** | A collection representing the allowed fuel grades that should be used within the authrorize parameters. |
| **IAuthorizeParameters** | The AuthorizeParameters is a data structure that contains the parameters that is used when authorizing a fuelling. |
| **IForecourtConfig** | The Forecourt Config interface is used to configure the forecourt using XML documents. The Xml document should conform to the ForecourtConfig.xsd schema. |
| **IForecourtControl** | The ForecourtControl object is the main root object to the Forecourtcontrol object hiearchy. It owns a list of pumps, and provides functionality to control the site. |
| **IFuelling** | Represents a fuelling. It provides properties to display Amount, Volume and so on for a fuelling. If the fuelling is reserved, the fuelling can be manipulated through the IReservedFuelling interface. |
| **IFuelPrice** | The fuel price represents the prices for one Fuel grade for the whole Forecourt. The prices are read Total fuel price for a fuelling is calculated as FuelPrice.BasePrice + FuelPrice.GeneralPriceChange + FuelPrice.AddPrices[ current price group] |
| **IFuelPriceAddPricePerPriceGroup** | Represents the Addprices for a FuelGrade object. |
| **IManualFuelDeliveryParameters** | Data structure that contains data for a manual fuel delivery registration. |
| **INozzle** | Represents a nozzle |
| **IPricePole** | Represents a price display on the forecourt. The price display is constructed of a series of price display segments, that each display the price for one fuel grade in one price group. |
| **IPricePoleSegment** | A display segment will display the configured unit price, FuelPrice [ PricePoleSegment..FuelGrade, PricePoleSegment..FuelGrade PriceGroup ]. The price cannot be set directly by the application, since there are law's and regulations about when Price pole and pump price can be updated, dependent on price increase or price decrease. Physical update of the PricePoleDisplay Segments, and pumps, are made automatically by the Forecourt server. when ActivateFuelPrices method in the Forecourt control object is called. |
| **IPump** | The IPump interface represents a logical fuel dispenser. It does only contain the methods that can be called without a pump reservation. When the pump is reserved, the IReservedPump interface is used, with an extended set of methods. |
| **ITank** | Interface to the representation of a physical fuel tank. The tanks are grouped in Tank groups, where several tanks are linked together. |
| **ITankGroup** | The Tank group interface is used to block and unblock fuellings with nozzles connected to any of the tanks in the tank group. |
| **ITankReading** | Data structure carrying the information of a physical tank reading. |

## Classes

| | |
|---|---|
| **AlarmEventArgs** | Event argument for an Alarm event. |
| **AllowedFuelGrades** | A collection representing the allowed fuel grades that should be used within the authrorize parameters. |
| **AuthorizeParameters** | The AuthorizeParameters is a data structure that contains the parameters that is used when authorizing a fuelling. |
| **ForecourtControlException** | The ForecourtControlException is the general exception that will be thrown from the implementations of ForecourtControllerInterface. |
| **ForecourtControlXml** | Xml serialization support class. |
| **FuelDeliveryEventArgs** | Data structure that contains data for a manual fuel delivery registration. |
| **FuelGradeOutOfRangeException** | Exception thrown from the AllowedFuelGrades class when trying to access a fuel grade that does not exist. |
| **FuellingData** | Fuelling data, contains serializable data from a fuelling. |
| **FuellingDataChangeEventArgs** | Event argument for a FuellingDataChanged event in the Pump object. |
| **FuellingStateChangeEventArgs** | Event argument for a FuellingState change event in a Pump object. |
| **ManualFuelDeliveryParameters** | Manual tank delivery data that is used when registering a manual delivery. |
| **NozzleStateChangeEventArgs** | Event argument for a PumpStateChange Event. |
| **PumpAccumulatorReading** | Data structure for one pump accumulator reading. |
| **PumpEventOccuredEventArgs** | Event argument for a PumpEvent, specifying that a certain event has occured on a pump. |
| **PumpStateChangeEventArgs** | Event argument for a PumpStateChange Event. |
| **SiteModeChangeEventArgs** | Event argument in the OnSiteModeChange event in ForecourtControl. |

### Enumerations

| | |
|---|---|
| **FuelDeliveryType** | Classifies the source of a fuel delivery information. |
| **FuellingState** | The possible states of a fuelling. |
| **FuellingType** | Type of a fuelling |
| **NozzleState** | The state of the nozzle. |
| **PresetType** | PresetType |
| **ProbeState** | Status of a Tank probe reading |
| **PumpAccumulatorReadingType** | Type of pump accumulator reading. |
| **PumpEventType** | Specifies the possible pump events that can be signalled using the IPump.SignalEvent. |
| **PumpState** | Pump state |
| **UnitOfMeasure** | Unit of measure used in the tank reading. |

## 9.1 Interfaces

### 9.1.1 Interface IAllowedFuelGrades

```
public interface IAllowedFuelGrades
```

**Summary**
A collection representing the allowed fuel grades that should be used within the authrorize parameters.

**Properties**

| | | |
|---|---|---|
| Count<br>`int` | R | Indicates the number of fuel grades. |
| Item<br>`bool` | R/W | Indexer returning if the fuel grade is allowed. |

### 9.1.2 Interface IAuthorizeParameters

```
public interface IAuthorizeParameters
```

**Summary**
The AuthorizeParameters is a data structure that contains the parameters that is used when authorizing a fuelling.

**Properties**

| | | |
|---|---|---|
| AllowedFuelGrade<br>`ForecourtControl.IAllowedFuelGrades` | R | Fuel grades allowed to use for the fuelling are set to true. |
| LockToReleaseClient<br>`bool` | R/W | The fuelling can only be reserved and set as paid by the releasing client if this property is set to true. |
| PresetType<br>`ForecourtControl.PresetType` | R/W | Specifies if the PresetValue should be regarded as a volume or an amount. |
| PresetValue<br>`decimal` | R/W | Maximum fuelling amount in domestic currency or volume. May be overridden by the configured maximum volume or amount in the pump controller. The lowest value will be used. |
| PriceGroup<br>`int` | R/W | Specifies the price group for the fuelling that should be used for the price calculation. |

### 9.1.3 Interface IForecourtConfig

```
public interface IForecourtConfig
```

**Summary**
The Forecourt Config interface is used to configure the forecourt using XML documents. The Xml document should conform to the ForecourtConfig.xsd schema.

**Methods**

**ReadConfiguration**
```
public string ReadConfiguration();
```
Reads the Forecourt control configuration and returns a XML structure.

| | |
|---|---|
| Return value | An Xml document as a string |

**WriteConfiguration**
```
public void WriteConfiguration(string configurationXml);
```
Writes the specified configuration XML to the forecourt control.

| | |
|---|---|
| *configurationXml* | A valid ForecourtConfig xml file. |

### 9.1.4 Interface IForecourtControl

```
public interface IForecourtControl
```

**Summary**
The ForecourtControl object is the main root object to the Forecourtcontrol object hiearchy. It owns a list of pumps, and provides functionality to control the site.

**Properties**

| | | |
|---|---|---|
| ClientId<br>`int` | R | ClientId in the communicationClientName in the communication to the Forecourt Controller. |
| ClientName<br>`string` | R | ClientName in the communication to the Forecourt Controller. |
| FuelPrices<br>`Collections.ObjectModel.ReadOnlyCollection {Wayne.ForecourtControl.IFuelPrice}` | R | The collection of Fuel prices. |
| PricePoles<br>`Collections.ObjectModel.ReadOnlyCollection {Wayne.ForecourtControl.IPricePole}` | R | The collection of Price poles. |
| Pumps<br>`Collections.ObjectModel.ReadOnlyCollection {Wayne.ForecourtControl.IPump}` | R | The collection of pump objects. |
| SiteMode<br>`int` | R | This property tells the operation mode of the site according to the pre-configured pump operation. Typically the configuration has been pre-defined with day, Night and rush hour operation modes, but this interface does not restrict which site mode that is used as day, night and rush-hour. |
| SiteOpened<br>`bool` | R | Tells whether the site is open or closed. |
| TankGroups<br>`Collections.ObjectModel.ReadOnlyCollection {Wayne.ForecourtControl.ITankGroup}` | R | The collection of Tank groups. |

**Methods**

**ActivateFuelPricesAsync**

```
public void
ActivateFuelPricesAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```

Activates the fuel prices, and triggers a new Fuel period when the prices has been activated. Note that it can take several minutes before all price signs and pumps have been updated. The FuelPrice reservation is released, so UnreserveFuelPricesAsync must not be called afterwards.

| | |
|---|---|
| *requestCompleted* | Callback delegate that will becalled on completion |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**ReserveFuelPricesAsync**
```
public void
ReserveFuelPricesAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```
Reserves the fuel prices so they can be changed by this client.

| | |
|---|---|
| *requestCompleted* | Callback delegate that will becalled on completion |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**SetSiteModeAsync**
```
public void SetSiteModeAsync(int siteMode,
EventHandler{Wayne.Lib.AsyncCompletedEventArgs} requestCompleted, object
userToken);
```
Sets the site mode. The pumps can be set in different modes for different site modes. This enables the option to have different day/night/rush modes of the station.

| | |
|---|---|
| *siteMode* | The new site mode. |
| *requestCompleted* | Delegate that will becalled on completion. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**SetSiteOpenedAsync**
```
public void SetSiteOpenedAsync(bool opened,
EventHandler{Wayne.Lib.AsyncCompletedEventArgs} requestCompleted, object
userToken);
```
Opens or closes the station.

| | |
|---|---|
| *opened* | True if the station should be opened, and false if it should be closed. |
| *requestCompleted* | Delegate that will be called on completion. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**UnreserveFuelPricesAsync**
```
public void
UnreserveFuelPricesAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```
Releases the fuel price reservation. If the fuel prices were no reserved, the function will still return success=true. Changes made to the fuel prices will be undone.

| | |
|---|---|
| *requestCompleted* | Callback delegate that will becalled on completion |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

### Events

**OnAlarm**
```
public EventHandler{Wayne.ForecourtControl.AlarmEventArgs} OnAlarm;
```
Event that is raised when an alarm is fired from forecourt controller.

**OnSiteModeChange**
```
public EventHandler{Wayne.ForecourtControl.SiteModeChangeEventArgs}
OnSiteModeChange;
```
Event Fired when the site mode, i.e. the Site mode or the Site open/close mode has changed.

### 9.1.5 Interface IFuelling

```
public interface IFuelling
```

**Summary**
Represents a fuelling. It provides properties to display Amount, Volume and so on for a fuelling. If the fuelling is reserved, the fuelling can be manipulated through the IReservedFuelling interface.

**Properties**

| | | |
|---|---|---|
| Amount<br>`decimal` | R | Filled amount in domestic currency value. |
| AuthorizationId<br>`long` | R | An identifiaction of the authorization that is originally returned in the async callback for the IReservePump.AuthorizeAsync. It is used to match the authorization with the fuelling completion. |
| CompletionDateTime<br>`DateTime` | R | The date time of when the fuelling was completed. |
| CompletionReason<br>`int` | R | A status code indicating what caused ending of the fuelling. 0=Ok1=Timeout2=BNT Timeout3=Disconnected4=BNT disconnected5=Stopped6=Volume or amount decreased7=Pulser error8=Pulser current error9=Zero fuelling10=No decimals set11=Price error12=Volume or amuont garbage13=Display error14=Checksum error |
| FuelGrade<br>`int` | R | Fuel grade used. |
| FuellingSequenceNumber<br>`int` | R | Fuelling sequence number is a unique number created for ever completed fuelling. Begins to count from 1 at system cold-start |
| FuelPeriodId<br>`int` | R | The Fuel period that the fuelling belongs to. |
| Nozzle<br>`ForecourtControl.INozzle` | R | The Nozzle object on which this fuelling was made |
| PresetType<br>`ForecourtControl.PresetType` | R | Specifies if PresetValue is Amount or Volume. |
| PresetValue<br>`decimal` | R | Preset value when the fuelling was released. |
| Price<br>`decimal` | R | Price used for the fuelling in domestic currency value. |
| PriceGroup<br>`int` | R | PriceGroup used. |
| Pump<br>`ForecourtControl.IPump` | R | Reference to the owning pump. |
| PumpAccumulator<br>`ForecourtControl.PumpAccumulatorReading` | R | Pump accumulator read after completed fuelling. May be null if pump accumulator reading not is supported. |

| | | |
|---|---|---|
| Quantity<br>`decimal` | R | Filled volume |
| ReservedBy<br>`int` | R | 0 if not reserved, else it contains the ClientId of the application that has reserved the fuelling. |
| State<br>`ForecourtControl.FuellingState` | R | State of the fuelling. |
| Type<br>`ForecourtControl.FuellingType` | R | Type of fuelling. |

**Methods**

**ReserveAsync**

```
public void ReserveAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
fuellingReserveCompleted, object userToken);
```

Reserves the fuelling for exclusive use. When the fuelling is successfully reserved, the ReservedBy property will be set to the ClientId of the reserving client.

| | |
|---|---|
| *fuellingReserveCompleted* | Callback delegate that will be called on completion. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback. |

**SetAsPaidAsync**

```
public void SetAsPaidAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```

Sets the fuelling to paid state, which means that it will no longer will be availiable in the Fuellings array.

| | |
|---|---|
| *requestCompleted* | Callback delegate that will be called on completion. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**TransferAsync**

```
public void TransferAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```

Changes the state of the fuelling to Transferred. If the fuelling is not already reserved, that is done implicitly, and must succeed before the transfer can succeed.

| | |
|---|---|
| *requestCompleted* | Callback delegate that will be called on completion. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**UndoTransferAsync**

```
public void UndoTransferAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```

Rolls back the transfer and unreserves the fuelling.

| | |
|---|---|
| *requestCompleted* | Callback delegate that will be called on completion. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**UnreserveAsync**

```
public void UnreserveAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```

Cancel fuelling lock.

| | |
|---|---|
| *requestCompleted* | |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

### 9.1.6 Interface IFuelPrice

```
public interface IFuelPrice
```

**Summary**
The fuel price represents the prices for one Fuel grade for the whole Forecourt. The prices are read Total fuel price for a fuelling is calculated as FuelPrice.BasePrice + FuelPrice.GeneralPriceChange + FuelPrice.AddPrices[ current price group]

**Properties**

| | | |
|---|---|---|
| BasePrice<br>`decimal` | R/W | Base price of the fuel grade |
| FuelGrade<br>`int` | R | The Fuel grade that this fuel price represents. |
| GeneralPriceDelta<br>`decimal` | R/W | A general price deviation from the base price that is used for all PriceGroups. |
| PriceGroupDelta<br>`ForecourtControl.IFuelPriceAddPricePerPriceGroup` | R | The Price deviation from the |
| Reserved<br>`bool` | R | Indicates if the Fuel price is reserved, and thus writable. |

### 9.1.7 Interface IFuelPriceAddPricePerPriceGroup

```
public interface IFuelPriceAddPricePerPriceGroup
```

**Summary**
Represents the Addprices for a FuelGrade object.

**Properties**

| | | |
|---|---|---|
| Item<br>`decimal` | R/W | Indexer that returns the AddPrice for a specific PriceGroup index (0-11) |

### 9.1.8 Interface IManualFuelDeliveryParameters

```
public interface IManualFuelDeliveryParameters
```

**Summary**
Data structure that contains data for a manual fuel delivery registration.

**Properties**

| | | |
|---|---|---|
| EndDateTime<br>`DateTime` | R/W | End date and time for the delivery. |
| PlannedQuantity<br>`decimal` | R/W | Optional. The quantity that was planned to deliver. |
| Quantity<br>`decimal` | R/W | The delivered quantity |

| | | |
|---|---|---|
| ReferenceNote<br>`string` | R/W | Optional. Note reference number entered by the truck driver. |
| SourcePlantInfo<br>`string` | R/W | Optional. Plant where the truck came from. Free format string. |
| StartDateTime<br>`DateTime` | R/W | Start date and time for the delivery. |
| TruckFuelTemperature<br>`decimal` | R/W | Optional. Temperature of the fuel in the truck. |

### 9.1.9 Interface INozzle

```
public interface INozzle
```

**Summary**
Represents a nozzle

**Properties**

| | | |
|---|---|---|
| FuelGrade<br>`int` | R | Indicates the fuel grade that is connected to this nozzle. |
| PrimaryTankGroupId<br>`int` | R | The primary tank group. |
| PrimaryTankGroupPercentage<br>`int` | R | Blend percentage drawn from the Primary TankGroup. The rest will be drawn from the secondary TankGroup. (100 % for non blending pumps.) |
| SecondaryTankGroupId<br>`int` | R | Pointer to the secondary tank group. Blend percentage is 100 % - PrimaryTankPercentage. |
| State<br>`ForecourtControl.NozzleState` | R | State of the Nozzle. |

**Methods**

**ReadPumpAccumulatorAsync**

```
public void ReadPumpAccumulatorAsync(EventHandler
{Wayne.Lib.AsyncCompletedEventArgs{Wayne.ForecourtControl.PumpAccumulatorReading}}
accumulatorsRead, object userToken);
```

Requests a momentary reading of the physical accumulators for the nozzle.

| | |
|---|---|
| *accumulatorsRead* | |
| *userToken* | |

### 9.1.10 Interface IPricePole

```
public interface IPricePole
```

**Summary**
Represents a price display on the forecourt. The price display is constructed of a series of price display segments, that each display the price for one fuel grade in one price group.

**Properties**

| | | |
|---|---|---|
| ConnectionState<br>`Lib.DeviceConnectionState` | R | The connection state of the price pole. |
| DisplaySegments | R | An array of the display |

| | | |
|---|---|---|
| `Collections.ObjectModel.ReadOnlyCollection {Wayne.ForecourtControl.IPricePoleSegment}` | | segments in the price pole. |
| Id<br>`int` | R | Price pole number |

**Events**

**OnConnectionStateChanged**
`public EventHandler{Wayne.Lib.ConnectionChangedEventArgs} OnConnectionStateChanged;`
Event fired when the connection state of the pricepole changes.

### 9.1.11 Interface IPricePoleSegment

`public interface IPricePoleSegment`

**Summary**
A display segment will display the configured unit price, FuelPrice [ PricePoleSegment..FuelGrade, PricePoleSegment..FuelGrade PriceGroup ]. The price cannot be set directly by the application, since there are law's and regulations about when Price pole and pump price can be updated, dependent on price increase or price decrease. Physical update of the PricePoleDisplay Segments, and pumps, are made automatically by the Forecourt server. when ActivateFuelPrices method in the Forecourt control object is called.

**Properties**

| | | |
|---|---|---|
| FuelGrade<br>`int` | R | The Fuel grade price that should be displayed |
| Id<br>`int` | R | The id of the price pole segment. |
| PriceGroup<br>`int` | R | The Price group price that should be displayed |

### 9.1.12 Interface IPump

`public interface IPump`

**Summary**
The IPump interface represents a logical fuel dispenser. It does only contain the methods that can be called without a pump reservation. When the pump is reserved, the IReservedPump interface is used, with an extended set of methods.

**Properties**

| | | |
|---|---|---|
| Blocked<br>`bool` | R | Indicates if the pump is blocked by a client |
| CapFuelGradeSelected<br>`bool` | R | Indicates if the pump is capable to supply information when a fuel grade is selected. |
| CapNozzleDetection<br>`bool` | R | True if pump protocol is capable to report nozzle in/out |
| CapSetLight<br>`bool` | R | True if the pump light on/off command is supported by the pump/protocol. |
| CapSuspendFuelling<br>`bool` | R | True if Suspend/Resume commands are supported by the pump/protocol. |

| | | |
|---|---|---|
| Connected<br>`bool` | R | Indicates if the pump is online to on the link. |
| CurrentFuelling<br>`ForecourtControl.IFuelling` | R | The running fuelling object. this should reflect what is shown on the pump display. |
| FuelGradeSelected<br>`bool` | R | Indicates if a valid fuel grade has been selected. Operation may be restricted by CapFuelGradeSelected for pump protocols not supporting this feature. |
| Fuellings<br>`Collections.ObjectModel.ReadOnlyCollection {Wayne.ForecourtControl.IFuelling}` | R | Fuelling collection holding the fuellings availiable for payment. |
| Id<br>`int` | R | Pump id, zero based pump number. The pump number that will be displayed is 1-based, so in this property, pump 1 will have Id 0. |
| Nozzles<br>`Collections.ObjectModel.ReadOnlyCollection {Wayne.ForecourtControl.INozzle}` | R | An array with Nozzles connected to this pump. |
| Open<br>`bool` | R | Indicates if the pump is open. |
| PriceGroup<br>`ForecourtControl.PriceGroup` | R | Price group that should be used to calculate the fuelling price. It will aslo set the unit price(s) present when the pump is idle. use SetPriceGroup to change the property. |
| ReservedBy<br>`int` | R | 0 if not reserved. If reserved then this contains the ClientId of the application that has reserved it (using Reserve command). |
| RunningFuellingUpdates<br>`bool` | R/W | Enable continous updates on the CurrentFuelling information during a fuelling. Events will be fired on OnFuellingDataChange when the filling data has changed, and current fuelling will be updated. |
| State<br>`ForecourtControl.PumpState` | R | State of the pump. |

**Methods**

**AuthorizeAsync**

```
public void AuthorizeAsync(ForecourtControl.IAuthorizeParameters
authorizeParameters,
EventHandler{Wayne.Lib.AsyncCompletedEventArgs{System.Int64}}
requestCompleted, object userToken);
```

Authorize pump for fuelling. The supplied AuthoriseParameters object contains the volume amount and grade restrictions of the release. The AsyncCompletedEventArgs will also contain a result (long) that will contain the AuthorizationId for the authorization. This can be matched with the IFuelling.AuthorizationId when the fuelling is running or is finished. This method may only be called when the pump is successfully reserved.

| | |
|---|---|
| *authorizeParameters* | Object that describes the rules for the authorization. |
| *requestCompleted* | Delegate that will be called after completion of the request. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**AuthorizeUpdateAsync**

```
public void AuthorizeUpdateAsync(ForecourtControl.IAuthorizeParameters
authorizeParameters, EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```

Update of the limits for an already authorised pump. This is the asynchronous version of the request, and will call the supplied delegate on completion. This method may only be called when the pump is successfully reserved.

| | |
|---|---|
| *authorizeParameters* | Object that describes the rules for the authorization. |
| *requestCompleted* | Delegate that will be called after completion of the request. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**ReserveAsync**

```
public void ReserveAsync(ForecourtControl.FuellingType fuellingType, byte
deviceId, EventHandler{Wayne.Lib.AsyncCompletedEventArgs} reservedCompleted,
object userToken);
```

Async version of Reserve()
IPump.ReserveAsync(Wayne.ForecourtControl.FuellingType,System.Byte,System.EventHandler {Wayne.Lib.AsyncCompletedEventArgs},System.Object)

| | |
|---|---|
| *fuellingType* | The fuelling type that the pump will be reserved for. |
| *deviceId* | The Device id that the pump will be reserved for. For example the terminal number if it is reserved for a specific terminal. |
| *reservedCompleted* | |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**ResumeAsync**

```
public void ResumeAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
resumeCompleted, object userToken);
```

Resumes a suspended fuelling

| | |
|---|---|
| *resumeCompleted* | Callback delegate that is called on completion. |
| *userToken* | User supplied object that will be returned in the suspendCompleted callback. |

**SetBlockedAsync**

```
public void SetBlockedAsync(bool blocked,
EventHandler{Wayne.Lib.AsyncCompletedEventArgs} requestCompleted, object
userToken);
```

Blocks or unblocks a pump for operation.

| | |
|---|---|
| *blocked* | True if the pump should be blocked, and false if it should be unblocked. |
| *requestCompleted* | Delegate that gets called when operation is completed |
| *userToken* | A user supplied object that will be returned in the requestCompleted |

| | |
|---|---|
| | callback |

**SetPriceGroupAsync**
`public void SetPriceGroupAsync(ForecourtControl.PriceGroup priceGroup, EventHandler{Wayne.Lib.AsyncCompletedEventArgs} requestCompleted, object userToken);`
Sets the Idle price group for the pump. This is the price group that will be used to calculate the fuelprice that is shown on the pump display.

| | |
|---|---|
| *priceGroup* | |
| *requestCompleted* | Delegate that gets called when operation is completed |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**SignalEventAsync**
`public void SignalEventAsync(ForecourtControl.PumpEventType eventType, EventHandler{Wayne.Lib.AsyncCompletedEventArgs} signalEventCompleted, object userToken);`
Signals that something regarding this pump has happened. The event will be signalled to all registered clients using the OnEventOccured event.

| | |
|---|---|
| *eventType* | |
| *signalEventCompleted* | |
| *userToken* | |

**StopAsync**
`public void StopAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs} requestCompleted, object userToken);`
Stops the current activity on the pump.

| | |
|---|---|
| *requestCompleted* | Delegate that gets called when operation is completed. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**SuspendAsync**
`public void SuspendAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs} suspendCompleted, object userToken);`
This command temporary suspends a running fuelling i.e. stops the pump motors. It may be possible to continue the fuelling again when teh Resume command is called but not all pump modles support Suspend / Resume handling. In this case the fuelling will be stopped without any possibility to resume operation.

| | |
|---|---|
| *suspendCompleted* | Callback delegate that is called on completion. |
| *userToken* | User supplied object that will be returned in the suspendCompleted callback. |

**UnauthorizeAsync**
`public void UnauthorizeAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs} requestCompleted, object userToken);`
Cancel of a fuelling authorization. This is the asynchronous version of the request, and will call the supplied delegate on completion. This method may only be called when the pump is successfully reserved.

| | |
|---|---|
| *requestCompleted* | Delegate that will be called after completion of the request. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**UnreserveAsync**
```
public void UnreserveAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs}
requestCompleted, object userToken);
```
Cancel pump reservation asynchronously. The reservation allows the the reservation owner to authorize the pump.

| | |
|---|---|
| *requestCompleted* | Delegate that will be called after completion of the request. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**Events**

**OnEventOccured**
```
public EventHandler{Wayne.ForecourtControl.PumpEventOccuredEventArgs}
OnEventOccured;
```
Event that is fired when a client has signalled an event using the SignalEventAsync method or from inside the forecourt controller.

**OnFuellingDataChange**
```
public EventHandler{Wayne.ForecourtControl.FuellingDataChangeEventArgs}
OnFuellingDataChange;
```
Fired when a fuelling is running and the volume and amount values has changed.

**OnFuellingStateChange**
```
public EventHandler{Wayne.ForecourtControl.FuellingStateChangeEventArgs}
OnFuellingStateChange;
```
Fired when the state or reservation of a fuelling has changed. Returns a handle affected fuelling.

**OnNozzleStateChange**
```
public EventHandler{Wayne.ForecourtControl.NozzleStateChangeEventArgs}
OnNozzleStateChange;
```
Fired when a Nozzle has been hooked in or out. Returns a handle to itself. Event firing is dependent on CapNozzleDetection

**OnStateChange**
```
public EventHandler{Wayne.ForecourtControl.PumpStateChangeEventArgs}
OnStateChange;
```
Fired when a pump state is changed

### 9.1.13 Interface ITank

```
public interface ITank
```

**Summary**
Interface to the representation of a physical fuel tank. The tanks are grouped in Tank groups, where several tanks are linked together.

**Properties**

| | | |
|---|---|---|
| CapPhysicalReading | R | Indicates if this tank has a probe for physical tank reading. |

| | | |
|---|---|---|
| `bool` | | |
| ConnectionState<br>`Lib.DeviceConnectionState` | R | Indicates the connection state of the tank probe. |
| Id<br>`int` | R | Id of the tank. |
| LatestPhysicalReading<br>`ForecourtControl.ITankReading` | R | The latest physical reading from the TIG |
| TankGroup<br>`ForecourtControl.ITankGroup` | R | The tank group this tank is associated with. |

**Methods**

**ReadAsync**
```
public void ReadAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs
{Wayne.ForecourtControl.ITankReading}} readingCompleted, object userToken);
```
Starts a physical tank reading if a physical probe is available.

| | |
|---|---|
| *readingCompleted* | |
| *userToken* | |

**RegisterManualTankDippingAsync**
```
public void RegisterManualTankDippingAsync(decimal tankLevel,
EventHandler{Wayne.Lib.AsyncCompletedEventArgs} manualTankDippingRegistered,
object userToken);
```
Used to register manual tank dipping from the application.

| | |
|---|---|
| *tankLevel* | |
| *manualTankDippingRegistered* | |
| *userToken* | |

**Events**

**OnConnectionStateChange**
```
public EventHandler{Wayne.Lib.ConnectionChangedEventArgs}
OnConnectionStateChange;
```
Event fired when the connection state of the tank changes

## 9.1.14 Interface ITankGroup

```
public interface ITankGroup
```

**Summary**
The Tank group interface is used to block and unblock fuellings with nozzles connected to any of the tanks in the tank group.

**Properties**

| | | |
|---|---|---|
| Blocked<br>`bool` | R | Indicates if all nozzles connected to tanks in this tank group are blocked. |
| FuelGrade<br>`int` | R | The fuel product the tank group contains. |

| | | |
|---|---|---|
| Id<br>`int` | R | Tank group Id |
| Tanks<br>`Collections.ObjectModel.ReadOnlyCollection {Wayne.ForecourtControl.ITank}` | R | The collection of physical tanks that is associated with this tank group. |

**Methods**

**BlockAsync**
```
public void BlockAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs} blockCompleted, object userToken);
```
Disable fuelling for all pump nozzles linked to this tank group.

| | |
|---|---|
| *blockCompleted* | |
| *userToken* | |

**RegisterManualDeliveryAsync**
```
public void RegisterManualDeliveryAsync(ForecourtControl.IManualFuelDeliveryParameters manualDeliveryParameters, EventHandler{Wayne.Lib.AsyncCompletedEventArgs} deliveryRegistrationCompleted, object userToken);
```
Register a manual fuel delivery from the application.

| | |
|---|---|
| *manualDeliveryParameters* | |
| *deliveryRegistrationCompleted* | |
| *userToken* | |

**UnblockAsync**
```
public void UnblockAsync(EventHandler{Wayne.Lib.AsyncCompletedEventArgs} unblockCompleted, object userToken);
```
Enable fuelling for all pump nozzles linked to this tank group.

| | |
|---|---|
| *unblockCompleted* | |
| *userToken* | |

**Events**

**OnFuelDelivery**
```
public EventHandler{Wayne.ForecourtControl.FuelDeliveryEventArgs} OnFuelDelivery;
```
Event that is fired when a fuel delivery is detected. It can be both manual and probe detected deliveries.

## 9.1.15 Interface ITankReading

```
public interface ITankReading
```

**Summary**
Data structure carrying the information of a physical tank reading.

**Properties**

| | | |
|---|---|---|
| DateTime<br>`DateTime` | R | Date and time when the phsical reading was made. |

| | | |
|---|---|---|
| FuelLevel<br>`decimal` | R | Fuel level, read by the probe. |
| FuelLevelUnit<br>`ForecourtControl.UnitOfMeasure` | R | Unit of measure used for the fuel level. |
| FuelTemperature<br>`decimal` | R | Fuel temperature, read by the probe. |
| NormalizedFuelLevel<br>`decimal` | R | Normalized fuel volume recalculated to the normalized temperature (15° C). |
| NormalizedFuelLevelSupplied<br>`bool` | R | Indicates if the normalized fuel volume is supplied. |
| State<br>`ForecourtControl.ProbeState` | R | Status of the probe reading. |
| TankId<br>`int` | R | Id of the Tank where the reading was made. |
| WaterLevel<br>`decimal` | R | Water level, read by the probe. |
| WaterLevelUnit<br>`ForecourtControl.UnitOfMeasure` | R | Unit of measure for the water level. |

## 9.2 Classes

### 9.2.1 Class AlarmEventArgs

```
public class AlarmEventArgs : EventArgs
```

**Summary**
Event argument for an Alarm event.

**Properties**

| | | |
|---|---|---|
| AlarmCategory<br>`int` | R/W | Category of the alarm |
| AlarmCode<br>`int` | R/W | Alarm code. |
| DebugText<br>`string` | R/W | Debug text for the alarm. |
| DeviceId<br>`int` | R | Device id for the device that the alarm was about. |
| DeviceType<br>`int` | R/W | Type of device. |

**Constructors**

```
public AlarmEventArgs(int deviceId, int deviceType, int alarmCode, int
alarmCategory, string debugText);
```
Initializes a new instance of the AlarmEventArgs class.

| | |
|---|---|
| *deviceId* | Device id for the device that the alarm is about |
| *deviceType* | Type of device |
| *alarmCode* | Alarm code. |

| | |
|---|---|
| *alarmCategory* | Category of the alarm. |
| *debugText* | Debug text. |

### 9.2.2 Class AllowedFuelGrades

```
public class AllowedFuelGrades : Object
```

**Summary**
A collection representing the allowed fuel grades that should be used within the authrorize parameters.

**Properties**

| | | |
|---|---|---|
| Count<br>`int` | R | Indicates the number of fuel grades. |
| Item<br>`bool` | R/W | Indexer returning if the fuel grade is allowed. |

### 9.2.3 Class AuthorizeParameters

```
public class AuthorizeParameters : Object
```

**Summary**
The AuthorizeParameters is a data structure that contains the parameters that is used when authorizing a fuelling.

**Properties**

| | | |
|---|---|---|
| AllowedFuelGrade<br>`ForecourtControl.AllowedFuelGrades` | R | Fuel grades allowed to use for the fuelling are set to true. |
| LockToReleaseClient<br>`bool` | R/W | The fuelling can only be reserved and set as paid by the releasing client if this property is set to true. |
| PresetType<br>`ForecourtControl.PresetType` | R/W | Specifies if the PresetValue should be regarded as a volume or an amount. |
| PresetValue<br>`decimal` | R/W | Maximum fuelling amount in domestic currency or volume. May be overridden by the configured maximum volume or amount in the pump controller. The lowest value will be used. |
| PriceGroup<br>`int` | R/W | Specifies the price group that should be used for the price calculation. |

**Constructors**

| |
|---|
| `public AuthorizeParameters();`<br>Initializes a new instance of the AuthorizeParameters class. |

### 9.2.4 Class ForecourtControlException

```
public class ForecourtControlException : Exception
```

**Summary**
The ForecourtControlException is the general exception that will be thrown from the implementations of ForecourtControllerInterface.

**Constructors**

| |
|---|
| `public ForecourtControlException();`<br>Initializes a new instance of the class |

| | |
|---|---|
| `public ForecourtControlException(string message);`<br>Initializes a new instance of the class | |
| *message* | |

| | |
|---|---|
| `public ForecourtControlException(string message, Exception inner);`<br>Initializes a new instance of the class | |
| *message* | |
| *inner* | |

### 9.2.5 Class ForecourtControlXml

```
abstract public class ForecourtControlXml : Object
```

**Summary**
Xml serialization support class.

**Fields**

| | |
|---|---|
| Ns<br>`string` | Namespace for Forecourt control. |

**Methods**

| |
|---|
| **AddSchemas**<br>`public void AddSchemas(Xml.Schema.XmlSchemaSet xmlSchemaSet);`<br>Adds the internal schemas to an xml schema set object. |
| *xmlSchemaSet* |

### 9.2.6 Class FuelDeliveryEventArgs

```
public class FuelDeliveryEventArgs : EventArgs
```

**Summary**
Data structure that contains data for a manual fuel delivery registration.

**Properties**

| | | |
|---|---|---|
| EndDateTime<br>`DateTime` | R | End date and time for the delivery. |
| PlannedQuantity<br>`decimal` | R | The volume that was planned to deliver. |
| Quantity<br>`decimal` | R | The delivered volume |
| ReferenceNote<br>`string` | R | Note reference number entered by the truck driver. |
| SourcePlantInfo<br>`string` | R | Plant where the truck came from. Free format string. |

| | | |
|---|---|---|
| StartDateTime<br>`DateTime` | R | Start date and time for the delivery. |
| TankGroupId<br>`int` | R | Id of the tank group that the delivery was made to. |
| TruckFuelTemperature<br>`decimal` | R | Temperature of the fuel in the truck. |
| Type<br>`ForecourtControl.FuelDeliveryType` | R | Defines how the delivery was registered. I.e detected from a tank probe or a manual registration. |

**Constructors**

```
public FuelDeliveryEventArgs(ForecourtControl.FuelDeliveryType type, int tankGroupId, DateTime startDateTime, DateTime endDateTime, decimal quantity, decimal plannedQuantity, decimal truckFuelTemperature, string sourcePlantInfo, string referenceNote);
```

| | |
|---|---|
| *type* | Defines how the delivery was registered. I.e detected from a tank probe or a manual registration. |
| *tankGroupId* | Id of the tank group that the delivery was made to. |
| *startDateTime* | Start date and time for the delivery. |
| *endDateTime* | End date and time for the delivery. |
| *quantity* | The delivered volume. |
| *plannedQuantity* | The volume that was planned to deliver. |
| *truckFuelTemperature* | Temperature of the fuel in the truck. |
| *sourcePlantInfo* | Plant where the truck came from. Free format string. |
| *referenceNote* | Note reference number entered by the truck driver. |

### 9.2.7 Class FuelGradeOutOfRangeException

```
public class FuelGradeOutOfRangeException : Exception
```

**Summary**
Exception thrown from the AllowedFuelGrades class when trying to access a fuel grade that does not exist.

**Constructors**

```
public FuelGradeOutOfRangeException();
```
Initializes an new instance of the class.

```
public FuelGradeOutOfRangeException(string message);
```
Initializes an new instance of the class.

| | |
|---|---|
| *message* | |

```
public FuelGradeOutOfRangeException(string message, Exception inner);
```
Initializes an new instance of the class.

| | |
|---|---|
| *message* | |
| *inner* | |

### 9.2.8 Class FuellingData

```
public class FuellingData : Object
```

**Summary**
Fuelling data, contains serializable data from a fuelling.

**Properties**

| | | |
|---|---|---|
| Amount<br>`decimal` | R | Fuelled amount |
| AuthorizationId<br>`long` | R | Authorization Id of the fuelling authorization. |
| CompletionDateTime<br>`DateTime` | R | Date and time when the fuelling completed. |
| CompletionReason<br>`int` | R | A status code signaling why the fuelling stopped. |
| FuelGrade<br>`int` | R | Fuelgrade that was used in the fuelling. |
| FuellingSequenceNumber<br>`int` | R | Fuelling sequence number is a unique number created for ever completed fuelling. Begins to count from 1 at system cold-start |
| FuelPeriodId<br>`int` | R | Fuel period that the fuelling belongs to. |
| NozzleId<br>`int` | R | Id of the nozzle that performed the fuelling |
| PresetType<br>`ForecourtControl.PresetType` | R | Indicates if the preset value contains an amount or quantity. |
| PresetValue<br>`decimal` | R | Preset amount or quantity |
| Price<br>`decimal` | R | Unit price of the fuelling. |
| PriceGroup<br>`ForecourtControl.PriceGroup` | R | Price group that was used to calculate the price in the fuelling. |
| PumpId<br>`int` | R | Id of the pump that performed the fuelling |
| Quantity<br>`decimal` | R | Fuelled quantity |
| Type<br>`ForecourtControl.FuellingType` | R | Type of the fuelling |

**Constructors**

| | |
|---|---|
| `public FuellingData(ForecourtControl.IFuelling fuelling);`<br>Initializes a new fuellingdata object using a ForecourtControl IFuelling object. | |
| *fuelling* | |

**Methods**

| **Deserialize**<br>`public ForecourtControl.FuellingData Deserialize(Xml.XmlElement fuellingDataElement);`<br>Recreates a fuelling data object from an Xml element. | |
|---|---|
| *fuellingDataElement* | |

| **WriteXml**<br>`public void WriteXml(Xml.XmlWriter writer, string prefix);`<br>Serializes the fuelling data. | |
|---|---|
| *writer* | |
| *prefix* | |

## 9.2.9 Class FuellingDataChangeEventArgs

```
public class FuellingDataChangeEventArgs : EventArgs
```

**Summary**
Event argument for a FuellingDataChanged event in the Pump object.

**Properties**

| | | |
|---|---|---|
| Amount<br>`decimal` | R | New amount |
| Fuelling<br>`ForecourtControl.IFuelling` | R | Fuelling which change has changed |
| Quantity<br>`decimal` | R | New Quantity |

**Constructors**

| `public FuellingDataChangeEventArgs(ForecourtControl.IFuelling fuelling, decimal amount, decimal quantity);`<br>Constructor | |
|---|---|
| *fuelling* | |
| *amount* | |
| *quantity* | |

**Methods**

## 9.2.10 Class FuellingStateChangeEventArgs

```
public class FuellingStateChangeEventArgs : EventArgs
```

**Summary**
Event argument for a FuellingState change event in a Pump object.

**Properties**

| | | |
|---|---|---|
| Fuelling<br>`ForecourtControl.IFuelling` | R | Fuelling which change has changed. If this parameter is NULL, then a fuelling have been removed from the pump. |

| | | |
|---|---|---|
| State<br>`ForecourtControl.FuellingState` | R | New Fuelling state |

**Constructors**

| |
|---|
| `public FuellingStateChangeEventArgs(ForecourtControl.IFuelling fuelling, ForecourtControl.FuellingState state);`<br>Constructor |
| *fuelling* |
| *state* |

**Methods**

## 9.2.11 Class ManualFuelDeliveryParameters

```
public class ManualFuelDeliveryParameters : Object
```

**Summary**
Manual tank delivery data that is used when registering a manual delivery.

**Properties**

| | | |
|---|---|---|
| EndDateTime<br>`DateTime` | R/W | End date and time for the delivery. |
| PlannedQuantity<br>`decimal` | R/W | Optional. The quantity that was planned to deliver. |
| Quantity<br>`decimal` | R/W | The delivered quantity |
| ReferenceNote<br>`string` | R/W | Optional. Note reference number entered by the truck driver. |
| SourcePlantInfo<br>`string` | R/W | Optional. Plant where the truck came from. Free format string. |
| StartDateTime<br>`DateTime` | R/W | Start date and time for the delivery. |
| TruckFuelTemperature<br>`decimal` | R/W | Optional. Temperature of the fuel in the truck. |

**Constructors**

| |
|---|
| `public ManualFuelDeliveryParameters();`<br>Initializes a new instance of the ManualFuelDeliveryParameters class. |

## 9.2.12 Class NozzleStateChangeEventArgs

```
public class NozzleStateChangeEventArgs : EventArgs
```

**Summary**
Event argument for a PumpStateChange Event.

**Properties**

| | | |
|---|---|---|
| Nozzle<br>`ForecourtControl.INozzle` | R | The Nozzle which state has changed. |

| | | |
|---|---|---|
| NozzleState<br>`ForecourtControl.NozzleState` | R | The new Nozzle state. |

**Constructors**

`public NozzleStateChangeEventArgs(ForecourtControl.INozzle nozzle, ForecourtControl.NozzleState nozzleState);`
Constructor

| | |
|---|---|
| *nozzle* | The Nozzle which state has changed |
| *nozzleState* | The new Nozzle state |

**Methods**

## 9.2.13 Class PumpAccumulatorReading

`public class PumpAccumulatorReading : Object`

**Summary**
Data structure for one pump accumulator reading.

**Properties**

| | | |
|---|---|---|
| Amount<br>`decimal` | R | Read amount |
| FuelPeriodId<br>`int` | R | Fuel period that the reading was made in. |
| Nozzle<br>`ForecourtControl.INozzle` | R | The nozzle that the reading was done for. |
| Pump<br>`ForecourtControl.IPump` | R | Pump that made the reading. |
| Quantity<br>`decimal` | R | Read quantity |
| Type<br>`ForecourtControl.PumpAccumulatorReadingType` | R | Type of accumulator reading. |

**Constructors**

`public PumpAccumulatorReading(ForecourtControl.IPump pump, ForecourtControl.INozzle nozzle, int fuelPeriodId, ForecourtControl.PumpAccumulatorReadingType type, decimal quantity, decimal amount);`
Constructor for the .Net version

| | |
|---|---|
| *pump* | |
| *nozzle* | |
| *fuelPeriodId* | |
| *type* | |
| *quantity* | |
| *amount* | |

| |
|---|
| `public PumpAccumulatorReading(ForecourtControl.Com.IPump pump, ForecourtControl.Com.INozzle nozzle, int fuelPeriodId, ForecourtControl.PumpAccumulatorReadingType type, decimal quantity, decimal amount);`<br>Constructor for the COM version |

| | |
|---|---|
| *pump* | |
| *nozzle* | |
| *fuelPeriodId* | |
| *type* | |
| *quantity* | |
| *amount* | |

### 9.2.14 Class PumpEventOccuredEventArgs

```
public class PumpEventOccuredEventArgs : EventArgs
```

**Summary**
Event argument for a PumpEvent, specifying that a certain event has occured on a pump.

**Properties**

| | | |
|---|---|---|
| EventType<br>`ForecourtControl.PumpEventType` | R | The type of the Event |

**Constructors**

| |
|---|
| `public PumpEventOccuredEventArgs(ForecourtControl.PumpEventType eventType);`<br>Initializes a new instance of the PunpEventOccuredEventArgs class. |

| | |
|---|---|
| *eventType* | The event type that occured. |

**Methods**

### 9.2.15 Class PumpStateChangeEventArgs

```
public class PumpStateChangeEventArgs : EventArgs
```

**Summary**
Event argument for a PumpStateChange Event.

**Properties**

| | | |
|---|---|---|
| Pump<br>`ForecourtControl.IPump` | R/W | The Pump whose state was changed. |
| PumpState<br>`ForecourtControl.PumpState` | R | The new Pump state. |

**Constructors**

| |
|---|
| `public PumpStateChangeEventArgs(ForecourtControl.IPump pump, ForecourtControl.PumpState pumpState);`<br>Constructor |

| | |
|---|---|
| *pump* | |

| | |
|---|---|
| *pumpState* | The new pump state |

**Methods**

### 9.2.16 Class SiteModeChangeEventArgs

```
public class SiteModeChangeEventArgs : EventArgs
```

**Summary**
Event argument in the OnSiteModeChange event in ForecourtControl.

**Properties**

| | | |
|---|---|---|
| SiteMode<br>`int` | R | Sets the site mode of the station. |
| SiteOpen<br>`bool` | R | Opens or closes the site. |

**Constructors**

| | |
|---|---|
| `public SiteModeChangeEventArgs(int siteMode, bool siteOpen);`<br>Constructor | |
| *siteMode* | |
| *siteOpen* | |

**Methods**

## 9.3 Enumerations

### 9.3.1 Enumeration FuelDeliveryType

**Summary**
Classifies the source of a fuel delivery information.

**Fields**

| | |
|---|---|
| Detected | The fuel delivery was detected from a tank probe. |
| Manual | The fuel delivery was manually registered. |

### 9.3.2 Enumeration FuellingState

**Summary**
The possible states of a fuelling.

**Fields**

| | |
|---|---|
| CurrentPumpData | Current pump data |
| Alive | Alive fuelling. |
| Fuelling | Fuelling |
| Transferred | The fuelling is transferred, and waiting to be paid. |
| Locked | Fuelling is locked |
| PayableTransaction | Fuelling is payable. |

| | |
|---|---|
| Unknown | Unknown |

### 9.3.3 Enumeration FuellingType

**Summary**
Type of a fuelling

**Fields**

| | |
|---|---|
| Unknown | Unknown fuelling type. |
| FullService | Full service fuelling. |
| Postpaid | Post paid fuelling . |
| Prepaid | Prepay fuelling |
| OptCardPaid | Outdoor Card fuelling (CT) |
| OptCashPaid | Outdoor Cash fuelling (BNT) |
| DetectedFromAccumulators | A fuelling that was detected from a difference in the pump accumulators. |
| TestFuelling | Test fuelling. |

### 9.3.4 Enumeration NozzleState

**Summary**
The state of the nozzle.

**Fields**

| | |
|---|---|
| Unknown | The state of the nozzle is unknown |
| In | The nozzle is resting. |
| Out | The nozzle is taken out. |

### 9.3.5 Enumeration PresetType

**Summary**
PresetType

**Fields**

| | |
|---|---|
| Unknown | Unspecified preset type |
| Amount | Preset is set by Amount. |
| Quantity | Preset is set by Quantity. |

### 9.3.6 Enumeration ProbeState

**Summary**
Status of a Tank probe reading

**Fields**

| | |
|---|---|
| Ok | The Probe reading was performed ok. |
| Failed | The probe reading failed. |

### 9.3.7 Enumeration PumpAccumulatorReadingType

**Summary**
Type of pump accumulator reading.

**Fields**

| | |
|---|---|
| Unknown | Unknown type. |
| AfterFuelling | After a fuelling completed. |
| AtConnect | At pump connect. |
| RequestedReading | When requested by application. |

### 9.3.8 Enumeration PumpEventType

**Summary**
Specifies the possible pump events that can be signalled using the IPump.SignalEvent.

**Fields**

| | |
|---|---|
| Stopped | The pump was stopped |
| CardRead | A card was read and blocks the pump. |
| CardAuthorized | A was authorized. |
| CardRejected | A card was rejected. |

### 9.3.9 Enumeration PumpState

**Summary**
Pump state

**Fields**

| | |
|---|---|
| Closed | Pump is closed. |
| Inoperative | Pump is inoperative |
| Idle | Pump is Idle |
| Calling | Pump is Calling for authorization |
| Authorized | Pump is authorised and ready to begin fuelling. |
| Fuelling | Pump is fuelling |
| Suspended | pump is Suspended |
| Unknown | Unknown state |

### 9.3.10 Enumeration UnitOfMeasure

**Summary**
Unit of measure used in the tank reading.

**Fields**

| | |
|---|---|
| Unknown | Unknown unit of measure. |
| Liters | Volume in litres |
| Millimeters | Height in millimeters |

# 10Namespace Wayne.ForecourtControl.Com

## Interfaces

| | |
|---|---|
| **IAlarmEventArgs** | Contains information about an alarm. |
| **IForecourtControl** | The ForecourtControl object is the main root object to the Forecourtcontrol object hiearchy. It owns a list of pumps, and provides functionality to control the site. |
| **IForecourtControlEvents** | Event interface for the IForecourtControl. Contains the event sink interface for the forecourt control object. |
| **IFuelDeliveryEventArgs** | Interface to Data structure that contains data for a manual fuel delivery registration. |
| **IFuelling** | Represents a fuelling. It provides properties to display Amount, Volume and so on for a fuelling. It does also give the possibilities to change the state of the fuelling and eventually get it removed when it is paid. |
| **IFuellingEvents** | Event interface for a IFuelling object. |
| **INozzle** | Represents a nozzle |
| **INozzleEvents** | Event interface for the INozzle. |
| **IPricePole** | Represents a price display on the forecourt. The price display is constructed of a series of price display segments, that each display the price for one fuel grade in one price group. |
| **IPricePoleEvents** | Event interface for an IPricePole class. |
| **IPricePoleSegment** | A display segment will display the configured unit price, FuelPrice [ PricePoleSegment..FuelGrade, PricePoleSegment..FuelGrade PriceGroup ]. The price cannot be set directly by the application, since there are law's and regulations about when Price pole and pump price can be updated, dependent on price increase or price decrease. Physical update of the PricePoleDisplay Segments, and pumps, are made automatically by the Forecourt server. when ActivateFuelPrices method in the Forecourt control object is called. |
| **IPump** | The IPump interface represents a logical fuel dispenser. It does only contain the methods that can be called without a pump reservation. When the pump is reserved, the IReservedPump interface is used, with an extended set of methods. |
| **IPumpAccumulatorReading** | Data structure for one pump accumulator reading. |
| **IPumpEvents** | Event interface for an IPump object. Contains the events that can be fired from a pump object. |
| **ITank** | Interface to the representation of a physical fuel tank. The tanks are grouped in Tank groups, where several tanks are linked together. |
| **ITankEvents** | Event interface for ITank |
| **ITankGroup** | The Tank group interface is used to block and unblock fuellings with nozzles connected to any of the tanks in the tank group. |
| **ITankGroupEvents** | Event interface for ITankGroup |

## Enumerations

| | |
|---|---|
| **DeviceConnectionState** | The state of the connection to a device. |

# 10.1 Interfaces

## 10.1.1 Interface IAlarmEventArgs

```
public interface IAlarmEventArgs
```

**Summary**
Contains information about an alarm.

**Properties**

| | | |
|---|---|---|
| AlarmCategory<br>`int` | R/W | Category of the alarm |
| AlarmCode<br>`int` | R/W | Alarm code. |
| DebugText<br>`string` | R/W | Debug text for the alarm. |
| DeviceId<br>`int` | R | Device id for the device that the alarm was about. |
| DeviceType<br>`int` | R/W | Type of device. |

## 10.1.2 Interface IForecourtControl

```
public interface IForecourtControl
```

**Summary**
The ForecourtControl object is the main root object to the Forecourtcontrol object hiearchy. It owns a list of pumps, and provides functionality to control the site.

**Properties**

| | | |
|---|---|---|
| ClientId<br>`int` | R | ClientId in the communicationClientName in the communication to the Forecourt Controller. |
| ClientName<br>`string` | R | ClientName in the communication to the Forecourt Controller. |
| ConnectionState<br>`ForecourtControl.Com.DeviceConnectionState` | R | Current connection state of the forecourt control. |
| FuelPrices<br>`ForecourtControl.IFuelPrice[]` | R | A list of the current fuel prices. The fuel prices can be modified when they are reserved through a call to ReserveFuelPricesAsync. The changes are committed through calling the ActivateFuelPricesAsync. |
| PricePoles<br>`ForecourtControl.Com.IPricePole[]` | R | A list of the price poles configured at the station. |
| Pumps<br>`ForecourtControl.Com.IPump[]` | R | The collection of pump objects. |
| SiteMode<br>`int` | R | This property tells the operation mode of the site according to the pre-configured pump operation. Typically the configuration has been pre-defined with day, Night and rush hour operation modes, but this |

| | | |
|---|---|---|
| | | interface does not restrict which site mode that is used as day, night and rush-hour. |
| SiteOpened<br>`bool` | R | Tells whether the site is open or closed. |
| TankGroups<br>`ForecourtControl.Com.ITankGroup[]` | R | A list of the tank groups configured on the station. |

**Methods**

**ActivateFuelPricesAsync**
```
public void ActivateFuelPricesAsync(object userToken);
```
Activates the fuel prices, and triggers a new Fuel period when the prices has been activated. Note that it can take several minutes before all price signs and pumps have been updated. The FuelPrice reservation is released, so UnreserveFuelPricesAsync must not be called afterwards.

| | |
|---|---|
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**Connect**
```
public void Connect(string connectionString);
```
Tries to connect the forecourt control using the specified connection string.

| | |
|---|---|
| *connectionString* | |

**Disconnect**
```
public void Disconnect();
```
Disconnects the forecourt control.

**ReserveFuelPricesAsync**
```
public void ReserveFuelPricesAsync(object userToken);
```
Reserves the fuel prices so they can be changed by this client.

| | |
|---|---|
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**SetSiteModeAsync**
```
public void SetSiteModeAsync(int siteMode, object userToken);
```
Sets the site mode. The pumps can be set in different modes for different site modes. This enables the option to have different day/night/rush modes of the station.

| | |
|---|---|
| *siteMode* | The new site mode. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**SetSiteOpenedAsync**
```
public void SetSiteOpenedAsync(bool opened, object userToken);
```
Opens or closes the station.

| | |
|---|---|
| *opened* | True if the station should be opened, and false if it should be closed. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**UnreserveFuelPricesAsync**
```
public void UnreserveFuelPricesAsync(object userToken);
```

| | |
|---|---|
| Releases the fuel price reservation. If the fuel prices were no reserved, the function will still return success=true. Changes made to the fuel prices will be undone. | |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

### 10.1.3 Interface IForecourtControlEvents

```
public interface IForecourtControlEvents
```

**Summary**
Event interface for the IForecourtControl. Contains the event sink interface for the forecourt control object.

**Methods**

**OnActivateFuelPricesCompleted**
```
public void
OnActivateFuelPricesCompleted(ForecourtControl.Com.IForecourtControl sender,
bool success, object userToken);
```
Event that will be invoked when a call to ActivateFuelPricesAsync has completed.

| | |
|---|---|
| *sender* | Sender object |
| *success* | True if the operation succeeded |
| *userToken* | The user token that was specified in the request. |

**OnAlarm**
```
public void OnAlarm(ForecourtControl.Com.IForecourtControl sender,
ForecourtControl.Com.IAlarmEventArgs e);
```

| | |
|---|---|
| *sender* | |
| *e* | |

**OnConnectionStateChange**
```
public void OnConnectionStateChange(ForecourtControl.Com.IForecourtControl
sender, ForecourtControl.Com.DeviceConnectionState connectionState);
```
Connection state to the Forecourt Control has changed.

| | |
|---|---|
| *sender* | |
| *connectionState* | |

**OnReserveFuelPricesCompleted**
```
public void
OnReserveFuelPricesCompleted(ForecourtControl.Com.IForecourtControl sender,
bool success, object userToken);
```
Event that will be invoked when a call to ReserveFuelPricesAsync has completed.

| | |
|---|---|
| *sender* | Sender object |
| *success* | True if the operation succeeded |
| *userToken* | The user token that was specified in the request. |

**OnSetSiteModeCompleted**
```
public void OnSetSiteModeCompleted(ForecourtControl.Com.IForecourtControl
```

| |
|---|
| `sender, bool success, object userToken);`<br>Event that will be invoked when a call to SetSiteModeAsync has completed. |

| | |
|---|---|
| *sender* | Sender object |
| *success* | True if the operation succeeded |
| *userToken* | The user token that was specified in the request. |

**OnSetSiteOpenedCompleted**
`public void OnSetSiteOpenedCompleted(ForecourtControl.Com.IForecourtControl sender, bool success, object userToken);`
Event that will be invoked when a call to SetSiteOpenendAsync has completed.

| | |
|---|---|
| *sender* | Sender object |
| *success* | True if the operation succeeded |
| *userToken* | The user token that was specified in the request. |

**OnSiteModeChange**
`public void OnSiteModeChange(ForecourtControl.Com.IForecourtControl sender, int siteMode, bool siteOpen);`
Site mode has changed. Either the site mode or the site open/closed status has changed.

| | |
|---|---|
| *sender* | |
| *siteMode* | |
| *siteOpen* | |

**OnUnreserveFuelPricesCompleted**
`public void OnUnreserveFuelPricesCompleted(ForecourtControl.Com.IForecourtControl sender, bool success, object userToken);`
Event that will be invoked when a call to UnreserveFuelPricesAsync has completed.

| | |
|---|---|
| *sender* | Sender object |
| *success* | True if the operation succeeded |
| *userToken* | The user token that was specified in the request. |

### 10.1.4 Interface IFuelDeliveryEventArgs

`public interface IFuelDeliveryEventArgs`

**Summary**
Interface to Data structure that contains data for a manual fuel delivery registration.

**Properties**

| | | |
|---|---|---|
| EndDateTime<br>`DateTime` | R | End date and time for the delivery. |
| PlannedQuantity<br>`decimal` | R | Optional. The volume that was planned to deliver. |
| Quantity<br>`decimal` | R | The delivered volume. |

| | | |
|---|---|---|
| ReferenceNote<br>`string` | R | Optional. Note reference number entered by the truck driver. |
| SourcePlantInfo<br>`string` | R | Optional. Plant where the truck came from. Free format string. |
| StartDateTime<br>`DateTime` | R | Start date and time for the delivery. |
| TankGroupId<br>`int` | R | Id of the tank group that the delivery was made to. |
| TruckFuelTemperature<br>`decimal` | R | Optional. Temperature of the fuel in the truck. |
| Type<br>`ForecourtControl.FuelDeliveryType` | R | Defines how the delivery was registered. I.e detected from a tank probe or a manual registration. |

### 10.1.5 Interface IFuelling

```
public interface IFuelling
```

**Summary**
Represents a fuelling. It provides properties to display Amount, Volume and so on for a fuelling. It does also give the possibilities to change the state of the fuelling and eventually get it removed when it is paid.

**Properties**

| | | |
|---|---|---|
| Amount<br>`decimal` | R | Filled amount in domestic currency value. |
| AuthorizationId<br>`int` | R | An identifiaction of the authorization that is originally returned in the async callback for the IReservePump.AuthorizeAsync. It is used to match the authorization with the fuelling completion. |
| CompletionDateTime<br>`DateTime` | R | The date time of when the fuelling was completed. |
| CompletionReason<br>`int` | R | A status code indicating what caused ending of the fuelling. 0=Ok1=Timeout2=BNT Timeout3=Disconnected4=BNT disconnected5=Stopped6=Volume or amount decreased7=Pulser error8=Pulser current error9=Zero fuelling10=No decimals set11=Price error12=Volume or amuont garbage13=Display error14=Checksum error |
| FuelGrade<br>`int` | R | Fuel grade used. |
| FuellingSequenceNumber<br>`int` | R | Fuelling sequence number is a unique number created for ever completed fuelling. Begins to count from 1 at system cold-start |
| FuelPeriodId<br>`int` | R | The Fuel period that the fuelling belongs to. |

| | | |
|---|---|---|
| Nozzle<br>`ForecourtControl.Com.INozzle` | R | The Nozzle object on which this fuelling was made |
| PresetType<br>`ForecourtControl.PresetType` | R | Specifies if PresetValue is Amount or Volume. |
| PresetValue<br>`decimal` | R | Preset value when the fuelling was released. |
| Price<br>`decimal` | R | Price used for the fuelling in domestic currency value. |
| PriceGroup<br>`int` | R | PriceGroup used. |
| Pump<br>`ForecourtControl.Com.IPump` | R | Reference to the owning pump. |
| PumpAccumulator<br>`ForecourtControl.Com.IPumpAccumulatorReading` | R | Pump accumulator read after completed fuelling. May be null if pump accumulator reading not is supported. |
| Quantity<br>`decimal` | R | Filled volume |
| ReservedBy<br>`int` | R | 0 if not reserved, else it contains the ClientId of the application that has reserved the fuelling. |
| State<br>`ForecourtControl.FuellingState` | R | State of the fuelling. |
| Type<br>`ForecourtControl.FuellingType` | R | Type of fuelling. |

**Methods**

**ReserveAsync**
```
public void ReserveAsync(object userToken);
```
Reserves the fuelling for exclusive use. When the fuelling is successfully reserved, the ReservedBy property will be set to the ClientId of the reserving client.

| | |
|---|---|
| *userToken* | A user supplied object that will be returned in the requestCompleted callback. |

**SetAsPaidAsync**
```
public void SetAsPaidAsync(object userToken);
```
Sets the fuelling to paid state, which means that it will no longer will be availiable in the Fuellings array.

| | |
|---|---|
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**TransferAsync**
```
public void TransferAsync(object userToken);
```
Changes the state of the fuelling to Transferred. If the fuelling is not already reserved, that is done implicitly, and must succeed before the transfer can succeed.

| | |
|---|---|
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**UndoTransferAsync**
```
public void UndoTransferAsync(object userToken);
```

| | |
|---|---|
| Rolls back the transfer and unreserves the fuelling. | |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

| | |
|---|---|
| **UnreserveAsync**<br>`public void UnreserveAsync(object userToken);`<br>Cancel fuelling lock. | |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

### 10.1.6 Interface IFuellingEvents

```
public interface IFuellingEvents
```

**Summary**
Event interface for a IFuelling object.

**Methods**

| | |
|---|---|
| **OnReserveCompleted**<br>`public void OnReserveCompleted(ForecourtControl.Com.IFuelling sender, bool success, object userToken);`<br>Event raised on completion of Reserve request. | |
| *sender* | Fuelling object that raised the event |
| *success* | True if the request succeeded. |
| *userToken* | User token specified when invoking the asynchronous request. |

| | |
|---|---|
| **OnSetAsPaidCompleted**<br>`public void OnSetAsPaidCompleted(ForecourtControl.Com.IFuelling sender, bool success, object userToken);`<br>Event raised on completion of SetAsPaid request. | |
| *sender* | Fuelling object that raised the event |
| *success* | True if the request succeeded. |
| *userToken* | User token specified when invoking the asynchronous request. |

| | |
|---|---|
| **OnTransferCompleted**<br>`public void OnTransferCompleted(ForecourtControl.Com.IFuelling sender, bool success, object userToken);`<br>Event raised on completion of Transfer request. | |
| *sender* | Fuelling object that raised the event |
| *success* | True if the request succeeded. |
| *userToken* | User token specified when invoking the asynchronous request. |

| | |
|---|---|
| **OnUndoTransferCompleted**<br>`public void OnUndoTransferCompleted(ForecourtControl.Com.IFuelling sender, bool success, object userToken);`<br>Event raised on completion of UndoTransfer request. | |
| *sender* | Fuelling object that raised the event |

| | |
|---|---|
| *success* | True if the request succeeded. |
| *userToken* | User token specified when invoking the asynchronous request. |

| **OnUnreserveCompleted**<br>`public void OnUnreserveCompleted(ForecourtControl.Com.IFuelling sender, bool success, object userToken);`<br>Event raised on completion of Unreserve request. | |
|---|---|
| *sender* | Fuelling object that raised the event |
| *success* | True if the request succeeded. |
| *userToken* | User token specified when invoking the asynchronous request. |

### 10.1.7 Interface INozzle

```
public interface INozzle
```

**Summary**
Represents a nozzle

**Properties**

| | | |
|---|---|---|
| FuelGrade<br>`int` | R | Indicates the fuel grade that is connected to this nozzle. |
| Id<br>`int` | R | Nozzle Id. |
| PrimaryTankGroupId<br>`int` | R | The primary tank group. |
| PrimaryTankGroupPercentage<br>`int` | R | Blend percentage drawn from the Primary TankGroup. The rest will be drawn from the secondary TankGroup. (100 % for non blending pumps.) |
| SecondaryTankGroupId<br>`int` | R | Pointer to the secondary tank group. Blend percentage is 100 % - PrimaryTankPercentage. |
| State<br>`ForecourtControl.NozzleState` | R | State of the Nozzle. |

**Methods**

| **ReadPumpAccumulatorAsync**<br>`public void ReadPumpAccumulatorAsync(object userToken);`<br>Requests a momentary reading of the physical accumulators for the nozzle. | |
|---|---|
| *userToken* | |

### 10.1.8 Interface INozzleEvents

```
public interface INozzleEvents
```

**Summary**
Event interface for the INozzle.

**Methods**

| **OnReadPumpAccumulatorCompleted** |
|---|

| |
|---|
| `public void OnReadPumpAccumulatorCompleted(ForecourtControl.Com.INozzle sender, ForecourtControl.Com.IPumpAccumulatorReading reading, object userToken);`<br>Response event for the ReadPumpAccumulators request. |

| | |
|---|---|
| *sender* | The nozzle object that fired the event. |
| *reading* | The pump accumulator reading data. |
| *userToken* | User token that was specified in the request. |

### 10.1.9 Interface IPricePole

`public interface IPricePole`

**Summary**
Represents a price display on the forecourt. The price display is constructed of a series of price display segments, that each display the price for one fuel grade in one price group.

**Properties**

| | | |
|---|---|---|
| ConnectionState<br>`ForecourtControl.Com.DeviceConnectionState` | R | The connection state of the price pole. |
| DisplaySegments<br>`ForecourtControl.Com.IPricePoleSegment[]` | R | An array of the display segments in the price pole. |
| Id<br>`int` | R | Price pole number |

### 10.1.10 Interface IPricePoleEvents

`public interface IPricePoleEvents`

**Summary**
Event interface for an IPricePole class.

**Methods**

| |
|---|
| **OnConnectionStateChange**<br>`public void OnConnectionStateChange(ForecourtControl.Com.IPricePole sender, ForecourtControl.Com.DeviceConnectionState connectionState);`<br>Connection state of the Price pole has changed. |

| | |
|---|---|
| *sender* | |
| *connectionState* | |

### 10.1.11 Interface IPricePoleSegment

`public interface IPricePoleSegment`

**Summary**
A display segment will display the configured unit price, FuelPrice [ PricePoleSegment..FuelGrade, PricePoleSegment..FuelGrade PriceGroup ]. The price cannot be set directly by the application, since there are law's and regulations about when Price pole and pump price can be updated, dependent on price increase or price decrease. Physical update of the PricePoleDisplay Segments, and pumps, are made automatically by the Forecourt server. when ActivateFuelPrices method in the Forecourt control object is called.

**Properties**

| | | |
|---|---|---|
| FuelGrade<br>`int` | R | The Fuel grade price that should be displayed |
| Id<br>`int` | R | The id of the price pole segment. |
| PriceGroup<br>`int` | R | The Price group price that should be displayed |

### 10.1.12 Interface IPump

```
public interface IPump
```

**Summary**
The IPump interface represents a logical fuel dispenser. It does only contain the methods that can be called without a pump reservation. When the pump is reserved, the IReservedPump interface is used, with an extended set of methods.

**Properties**

| | | |
|---|---|---|
| Blocked<br>`bool` | R | Indicates if the pump is blocked by a client |
| CapFuelGradeSelected<br>`bool` | R | Indicates if the pump is capable to supply information when a fuel grade is selected. |
| CapNozzleDetection<br>`bool` | R | True if pump protocol is capable to report nozzle in/out |
| CapSetLight<br>`bool` | R | True if the pump light on/off command is supported by the pump/protocol. |
| CapSuspendFuelling<br>`bool` | R | True if Suspend/Resume commands are supported by the pump/protocol. |
| Connected<br>`bool` | R | Indicates if the pump is online to on the link. |
| CurrentFuelling<br>`ForecourtControl.Com.IFuelling` | R | The running fuelling object. this should reflect what is shown on the pump display. |
| FuelGradeSelected<br>`bool` | R | Indicates if a valid fuel grade has been selected. Operation may be restricted by CapFuelGradeSelected for pump protocols not supporting this feature. |
| Fuellings<br>`ForecourtControl.Com.IFuelling[]` | R | Fuelling collection holding the fuellings availiable for payment. |
| Id<br>`int` | R | 0 based pump number for this instance. |
| Nozzles<br>`ForecourtControl.Com.INozzle[]` | R | An array with Nozzles connected to this pump. |
| Open<br>`bool` | R | Indicates if the pump is open. |
| PriceGroup<br>`int` | R | Idle price group used to calculate the prices to show on the pump when the pump is idle. When the pump is authorized, the pricegroup is specified again for that fuelling in the AuthorizeParameters. Use method SetPriceGroupAsync to change idle price group. |

| | | |
|---|---|---|
| ReservedBy<br>`int` | R | 0 if not reserved. If reserved then this contains the ClientId of the application that has reserved it (using Reserve command). |
| RunningFuellingUpdates<br>`bool` | R/W | Enable continous updates on the CurrentFuelling information during a fuelling. Events will be fired on OnFuellingDataChange when the filling data has changed, and current fuelling will be updated. |
| State<br>`ForecourtControl.PumpState` | R | State of the pump. |

**Methods**

**AuthorizeAsync**

```
public void AuthorizeAsync(ForecourtControl.IAuthorizeParameters
authorizeParameters, object userToken);
```

Authorize pump for fuelling. The supplied AuthoriseParameters object contains the volume amount and grade restrictions of the release. The AsyncCompletedEventArgs will also contain a result (long) that will contain the AuthorizationId for the authorization. This can be matched with the IFuelling.AuthorizationId when the fuelling is running or is finished.

| | |
|---|---|
| *authorizeParameters* | Object that describes the rules for the authorization. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**AuthorizeUpdateAsync**

```
public void AuthorizeUpdateAsync(ForecourtControl.IAuthorizeParameters
authorizeParameters, object userToken);
```

Update of the limits for an already authorised pump. This is the asynchronous version of the request, and will call the supplied delegate on completion.

| | |
|---|---|
| *authorizeParameters* | Object that describes the rules for the authorization. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**ReserveAsync**

```
public void ReserveAsync(ForecourtControl.FuellingType fuellingType, byte
deviceId, object userToken);
```

Async version of Reserve()
Com.IPump.ReserveAsync(Wayne.ForecourtControl.FuellingType,System.Byte,System.Object)

| | |
|---|---|
| *fuellingType* | The fuelling type that the pump will be reserved for. |
| *deviceId* | The Device id that the pump will be reserved for. For example the terminal number if it is reserved for a specific terminal. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**ResumeAsync**

```
public void ResumeAsync(object userToken);
```

Resumes a suspended fuelling

| | |
|---|---|
| *userToken* | User supplied object that will be returned in the suspendCompleted callback. |

**SetBlockedAsync**

```
public void SetBlockedAsync(bool blocked, object userToken);
```

| Blocks or unblocks a pump for operation. | |
|---|---|
| *blocked* | True if the pump should be blocked, and false if it should be unblocked. |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**SetPriceGroupAsync**
```
public void SetPriceGroupAsync(int priceGroup, object userToken);
```
Sets the Idle price group for the pump. This is the price group that will be used to calculate the fuelprice that is shown on the pump display.

| *priceGroup* | |
|---|---|
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

**SignalEventAsync**
```
public void SignalEventAsync(ForecourtControl.PumpEventType pumpEventType,
object userToken);
```
Signals that something regarding this pump has happened. The event will be signalled to all registered clients using the OnEventOccured event. When the operation completes, it is signalled through the event OnSignalEventCompleted.

| *pumpEventType* | Type of the event that occured. |
|---|---|
| *userToken* | User token to be returned in the completion event. |

**StopAsync**
```
public void StopAsync(object userToken);
```
Stops the current activity on the pump.

| *userToken* | A user supplied object that will be returned in the requestCompleted callback |
|---|---|

**SuspendAsync**
```
public void SuspendAsync(object userToken);
```
This command temporary suspends a running fuelling i.e. stops the pump motors. It may be possible to continue the fuelling again when teh Resume command is called but not all pump modles support Suspend / Resume handling. In this case the fuelling will be stopped without any possibility to resume operation.

| *userToken* | User supplied object that will be returned in the suspendCompleted callback. |
|---|---|

**UnauthorizeAsync**
```
public void UnauthorizeAsync(object userToken);
```
Cancel of a fuelling authorization. This command is only allowed after a successful call to Reserve() and Authorize. This is the asynchronous version of the request, and will call the supplied delegate on completion.

| *userToken* | A user supplied object that will be returned in the requestCompleted callback |
|---|---|

**UnreserveAsync**
```
public void UnreserveAsync(object userToken);
```
Cancel pump reservation asynchronously. When the request is completed, the supplied delegate will be called. After a call to this, the reference to the IReseredPump interface may not be used and should be

| | |
|---|---|
| unreferenced. | |
| *userToken* | A user supplied object that will be returned in the requestCompleted callback |

### 10.1.13 Interface IPumpAccumulatorReading

```
public interface IPumpAccumulatorReading
```

**Summary**
Data structure for one pump accumulator reading.

**Properties**

| | | |
|---|---|---|
| Amount<br>`decimal` | R | Read amount |
| FuelPeriodId<br>`int` | R | Fuel period that the reading was made in. |
| Nozzle<br>`ForecourtControl.Com.INozzle` | R | The nozzle that the reading was done for. |
| Pump<br>`ForecourtControl.Com.IPump` | R | Pump that made the reading. |
| Quantity<br>`decimal` | R | Read quantity |
| Type<br>`ForecourtControl.PumpAccumulatorReadingType` | R | Type of accumulator reading. |

### 10.1.14 Interface IPumpEvents

```
public interface IPumpEvents
```

**Summary**
Event interface for an IPump object. Contains the events that can be fired from a pump object.

**Methods**

**OnAuthorizeCompleted**
```
public void OnAuthorizeCompleted(ForecourtControl.Com.IPump sender, bool
success, int authorizationId, object userToken);
```
Event invoked when an Authorize request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *authorizationId* | Unique Id that identifies the authorization. Used to match an authorization to its fuelling later. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnAuthorizeUpdateCompleted**
```
public void OnAuthorizeUpdateCompleted(ForecourtControl.Com.IPump sender,
bool success, object userToken);
```
Event invoked when an AuthorizeUpdate request has completed.

| | |
|---|---|
| *sender* | |

| | |
|---|---|
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnEventOccured**
```
public void OnEventOccured(ForecourtControl.Com.IPump sender,
ForecourtControl.PumpEventType eventType);
```
Event that is fired when a client has signalled an event using the SignalEventAsync method or from inside the forecourt controller.

| | |
|---|---|
| *sender* | |
| *eventType* | Type of the event that occured. |

**OnFuellingDataChanged**
```
public void OnFuellingDataChanged(ForecourtControl.Com.IPump sender,
ForecourtControl.Com.IFuelling fuelling, decimal amount, decimal quantity);
```
Event fired when the fuelling data for a fuelling ( current fuelling ) changes.

| | |
|---|---|
| *sender* | Pump object that fired the event. |
| *fuelling* | The fuelling which data changed. |
| *amount* | The new amount. |
| *quantity* | The new quantity. |

**OnFuellingStateChanged**
```
public void OnFuellingStateChanged(ForecourtControl.Com.IPump sender,
ForecourtControl.Com.IFuelling fuelling, ForecourtControl.FuellingState
newState);
```
Event fired when a fuelling's state has changed.

| | |
|---|---|
| *sender* | Pump object that fired the event. |
| *fuelling* | The fuelling which state changed. |
| *newState* | New state of the fuelling. |

**OnNozzleStateChanged**
```
public void OnNozzleStateChanged(ForecourtControl.Com.IPump sender,
ForecourtControl.Com.INozzle nozzle, ForecourtControl.NozzleState
newNozzleState);
```
Event fired when a nozzle state changes.

| | |
|---|---|
| *sender* | |
| *nozzle* | |
| *newNozzleState* | |

**OnReserveCompleted**
```
public void OnReserveCompleted(ForecourtControl.Com.IPump sender, bool
success, object userToken);
```
Event invoked when a Reserve request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |

| | |
|---|---|
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnResumeCompleted**
```
public void OnResumeCompleted(ForecourtControl.Com.IPump sender, bool
success, object userToken);
```
Event invoked when a Resume request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnSetBlockedCompleted**
```
public void OnSetBlockedCompleted(ForecourtControl.Com.IPump sender, bool
success, object userToken);
```
Event invoked when a SetBlocked request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnSetPriceGroupCompleted**
```
public void OnSetPriceGroupCompleted(ForecourtControl.Com.IPump sender, bool
success, object userToken);
```
Event invoked when a SetPriceGroup request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnSignalEventCompleted**
```
public void OnSignalEventCompleted(ForecourtControl.Com.IPump sender, bool
success, object userToken);
```
Event invoked when an SignalEvent request completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnStateChange**
```
public void OnStateChange(ForecourtControl.Com.IPump sender,
ForecourtControl.PumpState newPumpState);
```
Event fired when the pump state has changed.

| | |
|---|---|
| *sender* | The object that fired the event. |
| *newPumpState* | The new pump state. |

**OnStopCompleted**
```
public void OnStopCompleted(ForecourtControl.Com.IPump sender, bool success,
object userToken);
```
Event invoked when a Stop request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnSuspendCompleted**
```
public void OnSuspendCompleted(ForecourtControl.Com.IPump sender, bool success, object userToken);
```
Event invoked when a Suspend request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnUnauthorizeCompleted**
```
public void OnUnauthorizeCompleted(ForecourtControl.Com.IPump sender, bool success, object userToken);
```
Event invoked when an Unauthorize request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnUnreserveCompleted**
```
public void OnUnreserveCompleted(ForecourtControl.Com.IPump sender, bool success, object userToken);
```
Event invoked when an Unreserve request has completed.

| | |
|---|---|
| *sender* | |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

### 10.1.15 Interface ITank

```
public interface ITank
```

**Summary**
Interface to the representation of a physical fuel tank. The tanks are grouped in Tank groups, where several tanks are linked together.

**Properties**

| | | |
|---|---|---|
| CapPhysicalReading<br>`bool` | R | Indicates if this tank has a probe for physical tank reading. |
| ConnectionState<br>`ForecourtControl.Com.DeviceConnectionState` | R | Indicates the connection state of the tank probe. |
| Id<br>`int` | R | Id of the tank. |
| LatestPhysicalReading<br>`ForecourtControl.ITankReading` | R | The latest physical reading from the TIG |

| | | |
|---|---|---|
| TankGroup<br>`ForecourtControl.Com.ITankGroup` | R | The tank group this tank is associated with. |

**Methods**

**ReadAsync**
```
public void ReadAsync(object userToken);
```
Starts a physical tank reading if a physical probe is available.

| | |
|---|---|
| *userToken* | |

**RegisterManualTankDippingAsync**
```
public void RegisterManualTankDippingAsync(decimal tankLevel, object
userToken);
```
Used to register manual tank dipping from the application.

| | |
|---|---|
| *tankLevel* | |
| *userToken* | |

### 10.1.16 Interface ITankEvents

```
public interface ITankEvents
```

**Summary**
Event interface for ITank

**Methods**

**OnConnectionStateChanged**
```
public void OnConnectionStateChanged(ForecourtControl.Com.ITank sender,
ForecourtControl.Com.DeviceConnectionState connectionState);
```
Event fired when the connection state of the tank changes

| | |
|---|---|
| *sender* | Tank object that raised the event. |
| *connectionState* | The new connection state of the tank. |

**OnReadCompleted**
```
public void OnReadCompleted(ForecourtControl.Com.ITank sender, bool success,
ForecourtControl.ITankReading tankReading, object userToken);
```
Event invoked when the ReadAsync request has completed.

| | |
|---|---|
| *sender* | Tank object that raised the event. |
| *success* | Indicates if the reading was performed ok. |
| *tankReading* | The tank reading that was created due to the read request. If success=false, tankreading will be null. |
| *userToken* | The user token that was specified in the request. |

**OnRegisterManualTankDippingCompleted**
```
public void OnRegisterManualTankDippingCompleted(ForecourtControl.Com.ITank
sender, bool success, object userToken);
```
Event invoked when the RegisterManualTankDippingAsync request has completed.

| | |
|---|---|
| *sender* | Tank object that raised the event. |
| *success* | True if the registration succeeded. |

| | |
|---|---|
| *userToken* | The user token that was specified in the request. |

### 10.1.17 Interface ITankGroup

```
public interface ITankGroup
```

**Summary**
The Tank group interface is used to block and unblock fuellings with nozzles connected to any of the tanks in the tank group.

**Properties**

| | | |
|---|---|---|
| Blocked<br>`bool` | R | Indicates if all nozzles connected to tanks in this tank group are blocked. |
| Id<br>`int` | R | Tank group Id |
| Tanks<br>`ForecourtControl.Com.ITank[]` | R | The collection of physical tanks that is associated with this tank group. |

**Methods**

**BlockAsync**
```
public void BlockAsync(object userToken);
```
Disable fuelling for all pump nozzles linked to this tank group.

| | |
|---|---|
| *userToken* | |

**RegisterManualDeliveryAsync**
```
public void
RegisterManualDeliveryAsync(ForecourtControl.IManualFuelDeliveryParameters
manualDeliveryParameters, object userToken);
```
Register a manual fuel delivery from the application.

| | |
|---|---|
| *manualDeliveryParameters* | The parameters for the manual delivery. |
| *userToken* | |

**UnblockAsync**
```
public void UnblockAsync(object userToken);
```
Enable fuelling for all pump nozzles linked to this tank group.

| | |
|---|---|
| *userToken* | |

### 10.1.18 Interface ITankGroupEvents

```
public interface ITankGroupEvents
```

**Summary**
Event interface for ITankGroup

**Methods**

**OnBlockCompleted**
```
public void OnBlockCompleted(ForecourtControl.Com.ITankGroup sender, bool
success, object userToken);
```
Event invoked when a Block request has completed.

| | |
|---|---|
| *sender* | The TankGroup object where the method was invoked. |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnFuelDelivery**
```
public void OnFuelDelivery(ForecourtControl.Com.ITankGroup sender,
ForecourtControl.Com.IFuelDeliveryEventArgs e);
```
Event that is raised when a fuel delivery is detected. It can be both manual and probe detected deliveries.

| | |
|---|---|
| *sender* | |
| *e* | |

**OnRegisterManualDeliveryCompleted**
```
public void OnRegisterManualDeliveryCompleted(ForecourtControl.Com.ITankGroup
sender, bool success, object userToken);
```
Event invoked when a RegisterManualDelivery request has completed.

| | |
|---|---|
| *sender* | The TankGroup object where the method was invoked. |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

**OnUnblockCompleted**
```
public void OnUnblockCompleted(ForecourtControl.Com.ITankGroup sender, bool
success, object userToken);
```
Event invoked when a Unblock request has completed.

| | |
|---|---|
| *sender* | The TankGroup object where the method was invoked. |
| *success* | True if the request succeeded. |
| *userToken* | Token object that was supplied in the asynchronous request. |

## 10.2 Enumerations

### 10.2.1 Enumeration DeviceConnectionState

**Summary**
The state of the connection to a device.

**Fields**

| | |
|---|---|
| Unknown | Unknown state of the connection. |
| Disconnected | Device is not connected. |
| Connecting | Trying to connect to device. |
| Connected | Connected to device. |
| Disconnecting | Disconnecting from device |