|  |  |
| --- | --- |
|  | Mitchell1 Information Systems  14145 Danielson Street  Poway, CA 92064 |

Mitchell1 ICatalog Software Development Kit

*Mitchell1’s Extension for Creating Shop Manager Catalogs*

Mitchell 1 Shop Management Solutions

04/24/2009

Poway, California

*Version 0.25*

# Revision Notes

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Revision** | **Changes** | **By** |
| 09/01/2008 | 0.10 | Initial version of documentation | Joe Dean |
| 09/16/2008 | 0.20 | Interface Documentation | Ken Richards |
| 10/8/2008 |  | Adding UML Class diagrams | Yama Kamyar Ken Richards |
| 10/8/2008 |  | Fixed various errors (spelling and incorrect signatures) | John Howard |
| 1/22/2009 |  | Corrected documentation on Status property of parts. | John Howard |
| 4/2/2009 |  | Changed descriptions of ImageUp and ImageDown buttons to promote standard button look and feel. | John Howard |
| 4/24/09 | 0.25 | Updated documentation related to DeliveryMethod. DeliveryMethod is not currently supported by Manager SE and is reserved for future use. | John Howard |
| 5/7/09 | 0.26 | Changed Custom Exceptions section to describe standard Error Messages. | John Howard |
| 7/16/13 | 0.30 | Added IPartItem2 documentation along with ICatalogVersion | Ken Richards |
| 1/23/13 | 0.35 | Added ICartOrder documentation | Phil Castaneda |
| 5/9/14 | 0.36 | Added CatalogAttributes | Phil Castaneda |

# Table of Contents

General 5

Mitchell1 Catalog Driver 6

Custom Exceptions and Error Handling 7

Interfaces for Catalog to Implement 12

*ICatalogInfo* 13

*ICatalogVersion* 14

*ICatalog* 15

*IDeliveryMethod* 17

Interfaces that Catalog Uses 18

Common Helper Classes 35

# General

## Introduction

## Goals

This document is meant to assist in the development of a parts catalog to run from and interact with the Mitchell1 Manager SE products. Using this guide, a developer should be able to create a parts catalog which will be dynamically loaded into the host and allow the user to setup a vendor link (authentication), shop for parts, check pricing, and order parts.

## Scope

This document covers the Mitchell1 Catalog Software Development Kit which contains details on how to develop a parts catalog that includes vendor setup, shopping for parts, price check, and parts ordering.

The parts catalog should be written using Microsoft C# over the .NET framework. Ideally, development should be done using Microsoft Visual Studio with .NET framework 2.0 or later.

The host application is designed to persist information such as the vehicle qualifiers, vendor setup, and all selected and ordered parts. Vendor information is **not** passed back to host application.

This document does not specify how the catalog should list parts, search for parts or maintain a shopping cart. These details are determined by the catalog implementer.

## Applicability

Any parts catalog which is compatible with the Catalog Driver, will also be compatible with the major Manager SE and Shopkey Management SE applications. Mitchell1 requires the catalog developers to provide their program code to be validated and digitally signed for publishing. The Shop management application only allows properly signed parts catalogs to be used.

## Conventions

The following font conventions are used in this document:

|  |  |
| --- | --- |
| STYLE | MEANING |
| FillCart | Indicates program code or verbatim text |
| **AddLocation** | Indicates a method or property name |
| **ICartItem** | Indicates a struct, class, or interface name |

# Mitchell1 Catalog Driver

The Mitchell1 Catalog Driver is used to test the catalog development in place of the Mitchell1 ShopStream Manager software. It tests all the essential features of a parts catalog without requiring them to be digitally signed. Only catalogs that have been validated and digitally signed by Mitchell1 can be run in the actual Manager SE product.

In general terms, the steps to test a parts catalog under Catalog Driver are:

1. Build your parts catalog which implements all of the required interfaces outlined in the SDK
2. Build and run the Catalog Driver application (in the **Driver** folder)
3. Select your catalog under the *Catalog* drop-down on the user interface
4. Push the vendor setup button to enter the authentication details
5. Click the “Go Shopping” button which appears on the Catalog Driver
6. Select parts and add them to your cart
7. Transfer the carts back to the Catalog Driver
8. Verify that all of the items from the shopping cart were transferred to Catalog Driver
9. Click on “Price Check” and initiate a check of prices for your transferred items
10. Verify that the prices, quantity, and availability were updated
11. Check the “Price Check” log to verify that the request and response are correct
12. Click on “Order Parts” and initiate an ordering of the parts you price checked
13. Verify that the status of the parts has changed to indicate the order status
14. Check the “Order Parts” log to verify that the request and response are correct
15. Check for all exception handling (authentication, communication, and general)

# Custom Exceptions and Error Handling

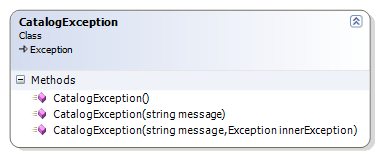
Catalogs should only throw exceptions listed in this document. Unexpected .NET exceptions will be caught by the host application and reported as a general exception. Methods in the catalog that display User Interface elements should handle their own errors and display messages as outlined in the Error Messages section below.

The supported exception types are:

****

## CatalogException

This exception should be thrown when an error occurs within the catalog which does not fit into the other catalog exception categories.

******

|  |  |
| --- | --- |
| CONSTRUCTOR | MEANING |
| CatalogException() | Initializes a new instance of the CatalogException class. No default message is defined to explain the exception. |
| CatalogException(**string** message) | Initializes a new instance of the CatalogException class with a specified error message. |
| CatalogException(**string** message, **Exception** innerException) | Initializes a new instance of the CatalogException class with a specified error message and a reference to the inner exception that is the cause of this exception. |

## CatalogConfigurationException

This exception should be thrown from GoShopping if configuration/credentials need to be re-entered, will cause Vendor Setup screen to appear instead of shopping.

## CatalogAuthenticationException

This exception should be thrown when the catalog cannot authenticate the vendor credentials.

******

|  |  |
| --- | --- |
| CONSTRUCTOR | MEANING |
| CatalogAuthenticationException() | Initializes a new instance of the CatalogAuthenticationException class. No default message is defined to explain the exception. |
| CatalogAuthenticationException(**string** message) | Initializes a new instance of the CatalogAuthenticationException class with a specified error message. |
| CatalogAuthenticationException(**string** message, **Exception** innerException) | Initializes a new instance of the CatalogAuthenticationException class with a specified error message and a reference to the inner exception that is the cause of this exception. |

CatalogCommunicationException

This exception should be thrown when an error occurs while communicating with the network.



|  |  |
| --- | --- |
| CONSTRUCTOR | MEANING |
| CatalogCommunicationException() | Initializes a new instance of the CatalogCommunicationException class. No default message is defined to explain the exception. |
| CatalogCommunicationException(**string** message) | Initializes a new instance of the CatalogCommunicationException class with a specified error message. |
| CatalogCommunicationException(**string** message, **Exception** innerException) | Initializes a new instance of the CatalogCommunicationException class with a specified error message and a reference to the inner exception that is the cause of this exception. |

Error Messages

The following error messages should be displayed for methods that display UI elements.

These messages should be displayed rather than throwing the related Exception.

For an error that occurs while communicating with the network, instead of throwing a CatalogCommunicationException, the following message should be displayed…

"No Internet connection, or remote server failed to respond."

The title bar should contain "Catalog Communication Error"

The information icon should display.

When the catalog cannot authenticate the vendor credentials, instead of throwing a CatalogAuthenticationException, the following message should be displayed…

"Please verify your vendor setup"

The title bar should contain "Catalog Authentication Error"

The information icon should display.

For general errors, instead of throwing a CatalogException, a message box with appropriate information should be displayed.

The title bar should contain "Catalog Error"

Interfaces for Catalog to Implement

The following interfaces must be implemented by a parts catalog developer.

Note that ICatalogVersion must be implemented on the same object as ICatalogInfo.

class CatalogInfo : ICatalogInfo, ICatalogVersion

{

// Implementation

}



# ICatalogInfo

This is the main interface that all Mitchell1 catalogs must implement. It contains catalog information that doesn't change based on the vendor configured in the host application.



|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Description | A description for the catalog (reserved for future use, possibly as a tool tip) |
| **string** DisplayName | The name of the catalog as displayed to the user |
| **Bitmap** ImageUp | The bitmap image displayed as the catalog button (normal state)  Button image should be 58 pixels wide by 28 pixels high with 24 bit color resolution.  To create this image, start with the BlankImageUp.bmp included in the SDK and paste a 50 pixel by 20 pixel image with a grey background {RGB (192,192,192)} into the BlankImageUp button at position (4,4) |
| **Bitmap** ImageDown | The bitmap image displayed as the pressed catalog button (pushed state)  Button image should be 58 pixels wide by 28 pixels high with 24 bit color resolution.  To create this image, start with the BlankImageDown.bmp included in the SDK and paste a 50 pixel by 20 pixel image with a grey background {RGB (192,192,192)} into the BlankImageDown button at position (6,6)  Note that the image on the down button is shifted 2 pixels right and 2 pixels down to give the appearance of a button being pressed. |

|  |
| --- |
| METHOD |
| **ICatalog** GetCatalog(**IVendor** vendor, **IVehicle** vehicle, **IHostData** hostData)  Method used to return the catalog object. |
| **bool** VendorSetup(**IVendor** vendor, **IHostData** hostData)  This method should display a dialog to allow the user to enter vendor settings such as username and password.  On Entry the Code and Name properties of vendor will have been set by the host application.  For new vendors, the vendor.Qualifier will be empty. For currently existing vendors, the vendor.Qualifier will contain a string as previously returned from VendorSetup.  On Exit the vendor.Qualifier property should contain custom vendor information (typically username and password coded as XML). The host application will persist this information and return it in subsequent method calls. |

## CatalogAttributes

This attribute allows a catalog class implementing ICatalogInfo to setup modified behavior rules in Manager SE.

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Default Value** | **Description** |
| TireCatalog | Boolean | false | When set to true, marks the ICatalogInfo catalog as being a Tire and Wheels only catalog. With this setting true or false, an IPartItem2 tire (IsTire=true) transferred from GoShopping is treated as a tire (category, fees, markup, etc).  When a non-tire (IsTire = false) is transferred and this flag is set to false (default), this part is treated as a normal part (no category, etc).  However, with TireCatalog marked as true, non-tire parts behave as wheels. This means, for example, that the default tire category will be applied.  **Note:** This flag does not change behavior of IPartItem parts; they will always be considered not a tire and not a wheel. |

*ICatalogVersion*

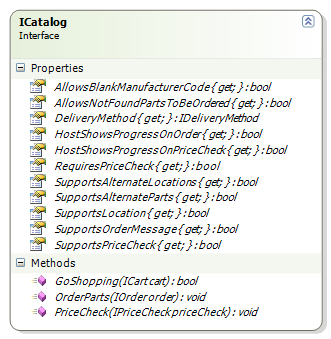
This interface defines a Version property to indicate the version of catalog interfaces being implemented.  
This interface should be implemented on the ICatalogInfo object for all new catalogs.  
Previous catalogs which do not implement this interface are assumed to have a Version of 2.0



|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **Version** Version | 2.0 is the base catalog version. 2.1 contains IPartItem2 with new Size and IsTire properties. |

## ICatalog

The ICatalog is the core component of the parts catalog. This interface is returned from the GetCatalog method in ICatalogInfo. All of the major features of a catalog are handled by the ICatalog including Go shopping, Price check, and Parts ordering.

******

|  |
| --- |
| METHOD |
| bool GoShopping(ICart cart);  This method should display a user interface to allow users to choose items. These items should be added to the cart as either an IPartItem, ILaborItem, or INoteItem. The cart argument may also implement ICartOrder in Manager SE versions 6.5 and up.  The return value indcates whether the method was successful. |
| void PriceCheck(IPriceCheck priceCheck);  This method should update the passed priceCheck object with pricing and availability information. The collection of parts is passed in as IPriceCheck. The catalog may suggest alternative parts for each part as well as listing more than one location for each part.  This method does not need to be implemented when the SupportsPriceCheck property is false. |
| void OrderParts(IOrder order);  This method should attempt to order the parts defined in the order object; The order object should be updated with information about the order. |

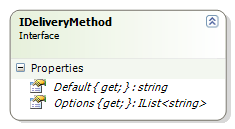
|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **bool** AllowsBlankManufacturerCode | Indicates whether the catalog allows an empty (undefined) manufacturer code for parts used in PriceCheck or OrderParts. |
| **bool** AllowsNotFoundPartsToBeOrdered | Indicates whether the catalog allows parts which were not found in the PriceCheck to be ordered. |
| **IDeliveryMethod** DeliveryMethod | Gets a list of Delivery Options and the Default so the host can allow the user to select one. If the catalog does not support delivery methods, this property should return null. (This property is currently ignored by Manager SE and is reserved for future use.) |
| **bool** HostShowsProgressOnOrder | Indicates whether the host application should display a window indicating progress while the OrderParts method is being executed. |
| **bool** HostShowsProgressOnPriceCheck | Indicates whether the host application should display a window indicating progress while the PriceCheck method is being executed. |
| **bool** RequiresPriceCheck | Indicates whether the host application should only allow ordering of parts that have been price checked. |
| **bool** SupportsAlternateLocations | Indicates whether the catalog supports alternate locations within the PriceCheck method. If true, the host application will allow the user to select from a list of Locations. |
| **bool** SupportsAlternateParts | Indicates whether the catalog supports alternate parts within the PriceCheck method. If true, the host application will allow the user to select from a list of AlternateParts. |
| **bool** SupportsLocation | Indicates whether the catalog can specify a location for a part. If true, the host application will display the part location. |
| **bool** SupportsOrderMessage | Indicates whether the catalog supports the OrderMessage property within the IOrder object. If true, the host application will allow the user to enter an order message that will be sent with the order. |
| **bool** SupportsPriceCheck | Indicates whether the catalog supports the PriceCheck method. If true, the host application will allow the user to perform a price check. If false, the host application will not allow the user to perform a price check. |

## IDeliveryMethod

The IDeliveryMethod interface represents a collection of delivery methods that a user may choose from when ordering parts. It also defines the default delivery method to use when the user does not select one explicitly. An instance of this class should be returned by the **DeliveryMethod** property in ICatalog.

This interface only needs to be implemented if the catalog supports delivery methods. If the catalog does not support delivery methods, the **DeliveryMethod** property in ICatalog should return null.

(The **DeliveryMethod** property is currently ignored by Manager SE and is reserved for future use.)



|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **IList<string>** Options | A list of delivery options a user may choose from when ordering parts. |
| **string** Default | The default delivery option. The host application will set this as the default. This string must exactly match one of the options returned by the Options property. |

Interfaces that Catalog Uses

This section contains all of the interfaces which the catalog may use.

## ICart

The ICart interface represents a shopping cart within the host application. It is the transfer mechanism which translates items from a catalog provider to the parts in the Catalog Driver. You will note that it is the first parameter to the **GoShopping** method.

## 

|  |  |
| --- | --- |
| METHOD | MEANING |
| ILaborItem NewLaborItem() | Create a new labor item for the shopping cart |
| INoteItem NewNoteItem() | Create a new note item for the shopping cart |
| IPartItem NewPartItem() | Create a new part item for the shopping cart |

Once a new item is created, it can be added to the Cart object using the **Add** method. The Cart object will be examined by the host application upon return from the **GoShopping** method (i.e. when the *Transfer* button is pushed).

Note that when you call NewPartItem(), you will actually get an object of type IPartItem2 which derives from IPartItem. You may cast the return value to IPartItem2 in order to use the two additional fields IsTire and Size.

## ICartOrder

The normal ICart object passed into GoShopping may also implement the ICartOrder interface which extends the existing functionality of ICart. This ICartOrder interface is used for ordering parts entirely within the GoShopping session (without the need to price check and order parts). However, this feature is only supported in newer versions of Manager so catalogs must check if the ICart argument in the GoShopping method implements this interface before using it.

The ICartOrder interface allows catalogs to offer users an alternative way of ordering parts – this does the work of the transfer, price check, and order parts calls.

**Note:** the previous behavior (using only ICart, transfer, price check, and order parts) can still be used. In addition, ICartOrder should only be used for already ordered parts. Non-ordered parts must remain in the ICart item list.

The order information added to the ICartOrder.Orders list will be used to generate required purchase order information as well as transfer parts to the repair order. Any non-ordered parts and/or non-part items must be added to the normal ICart parts list to be included on the repair order. Multiple purchase orders can be created at one time. This would happen, for example, when parts are selected from different branch/locations and then ordered through GoShopping.

## 

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| IList<IOrder> Orders | List of orders to be saved. Each order contains a list of parts for a specific branch (source warehouse) location. |

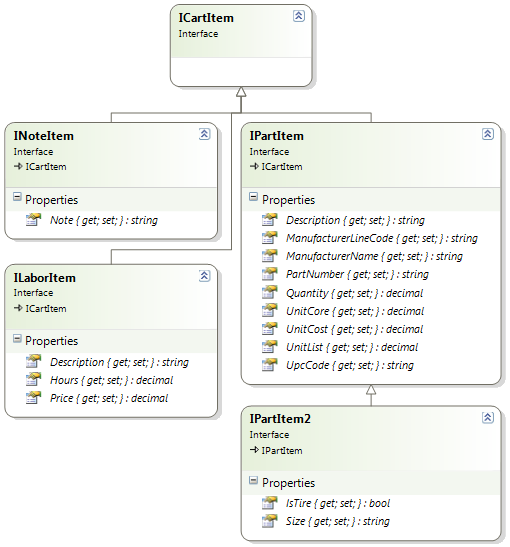
|  |  |
| --- | --- |
| METHOD | MEANING |
| IOrder NewOrder(string orderMessage, string deliveryOption) | Create a new order item to insert into Orders list. This order should only contain parts (from which a purchase order will be created in Manager) |
| IOrderPart NewOrderedPart() | Create a new ordered part item for inserting into the Order.Parts list. |

Within the GoShopping(…) method, check to see if the argument passed in implements the ICartOrder interface. If it does, use this to create new order(s) and then add the ordered parts to it. After the GoShopping method returns, Manager creates one or more purchase orders based on the contents of Orders list.

Items added to the ICart list will not be included in a purchase order. This retains the same behavior as previous GoShopping catalog operations. Items such as labor and notes still belong in the ICart list (along with non-ordered parts).

**Important:** Ordered items must not be placed into the ICart items list; otherwise, duplicated parts will be listed on the repair order. Ordered parts must only be listed in the ICartOrder.Orders parts list.*ICartItem*

This is the main base class for items which can go into a shopping cart. No classes should implement this interface directly. Instead, applications will use the derived interfaces: IPartItem, ILaborItem, or INoteItem.



## IPartItem

IPartItem represents a physical part that belongs to an ICart. This interface inherits from ICartItem. The **PartNumber** and the **ManufacturerLineCode** combined uniquely identify an individual part from the catalog.

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Description | Description for the part. |
| **string** ManufacturerLineCode | Manufacturer Line Code for the part. |
| **string** ManufacturerName | Manufacturer Name for the part. |
| **string** PartNumber | Part Number of part from catalog. |
| **decimal** Quantity | Quantity requested for the during GoShopping |
| **decimal** UnitCore | Cost of the core charge for the part. |
| **decimal** UnitCost | Cost for a single unit of this part |
| **decimal** UnitList | Suggested unit price for the part |
| **string** UpcCode | Universal Product Code for the part |

## IPartItem2

IPartItem2 represents a physical part that belongs to an ICart. This interface inherits from ICartItem and IPartItem. This represents a newer version of IPartItem that includes two additional fields **IsTire** and **Size**. The **PartNumber** and the **ManufacturerLineCode** combined uniquely identify an individual part from the catalog.

To use this newer part item, simply call ICart.NewPartItem() and cast the returned object instance to IPartItem2.

**Note: If a catalog uses IPartItem2 then the ICatalogInfo object should also implement ICatalogVersion with a Version value of 2.1 (or higher.)**

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Description | Description for the part. |
| **string** ManufacturerLineCode | Manufacturer Line Code for the part. |
| **string** ManufacturerName | Manufacturer Name for the part. |
| **string** PartNumber | Part Number of part from catalog. |
| **decimal** Quantity | Quantity requested for the during GoShopping |
| **decimal** UnitCore | Cost of the core charge for the part. |
| **decimal** UnitCost | Cost for a single unit of this part |
| **decimal** UnitList | Suggested unit price for the part |
| **string** UpcCode | Universal Product Code for the part |
| **bool** IsTire | Determines if the part is a tire or not |
| **string** Size | The size of the part (usually a tire size like “255x55 r17”) |

## ILaborItem

ILaborItem represents a labor work item that belongs to an ICart. This interface inherits from ICartItem. Labor items are created by calling the **NewLaborItem** method on an ICart and then added to the ICart using the **Add** method.

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Description | Description of the labor to be performed |
| **decimal** Hours | Estimated time that this labor item will take to complete |
| **decimal** Price | The suggested price to complete this labor item. If this property is 0, then the host application will calculate the price based on the Hours submitted and host’s labor rate. |

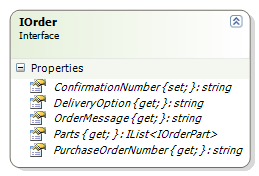
## INoteItem

INoteItem represents a note or comment that is added to the shopping cart. Note items should be added just before the IPartItem or the ILaborItem that they annotate. An INoteItem is not figured into the total cost for a repair order or any other billing item.

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Note | A note or comment to be placed on a repair order |

## IOrder

An IOrder interface represents a catalog order made through the host application. It contains a list of IOrderPart along with various other properties which uniquely identify the order. An instance of this type is passed as the first parameter to the **OrderParts** method in ICatalog.

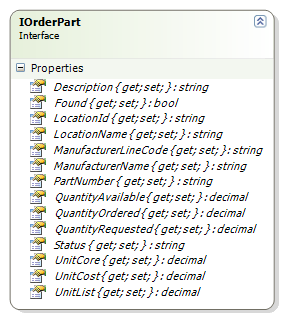
******

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** ConfirmationNumber | A confirmation number for the entire order used to verify the order was successful |
| **string** DeliveryOption | User selected delivery option from the **DeliveryMethod** property on ICatalog. (This property is currently ignored by Manager SE and is reserved for future use.) |
| **string** OrderMessage | User entered message sent to the catalog along with the order. |
| **IList<IOrderPart>** Parts | An ordered list of parts the user requested for ordering. These parts will be updated with the status once ordering is done. |
| **string** PurchaseOrderNumber | Purchase Order Number as determined by host application. |

## IOrderPart

The IOrderPart interface represents an individual line item found in a parts order (IOrder). Unlike the ICartItem, an order does not deal with notes or labor. Only physical parts are manipulated when performing parts ordering.

Together, the **ManufacturerLineCode** and the **PartNumber** uniquely identify a part in the order.

******

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Description | Description for the part to order. (This property should not be updated unless it is passed in as a blank string) |
| **bool** Found | Indicates whether the catalog recognize this part. (Set by the catalog) |
| **string** LocationId | Id of the location (Set by the host application as returned from **PriceCheck**). Optional based on whether a catalog supports **Locations.** (Max length 8 characters) |
| **string** LocationName | Name of the location. (Set by the host application as returned from a **PriceCheck**) Optional based on whether a catalog supports **Locations.** (Max length 20 characters) |
| **string** ManufacturerLineCode | Manufacturer Line Code of part to order. (Set by the host application) |
| **string** ManufacturerName | Manufacturer Name of part to order. |
| **string** PartNumber | Part Number of part to order. (Set by the host application) |
| **decimal** QuantityAvailable | Quantity of this part item available at this location. (Set by the catalog). |
| **decimal** QuantityOrdered | Quantity of part ordered. (Set by the catalog) |
| **decimal** QuantityRequested | Quantity of part requested to be ordered. (Set by the host application) |
| **string** Status | Custom Status of a processed part. Reserved for future use. |
| **decimal** UnitCore | Cost of a single core item. (Set by the catalog). |
| **decimal** UnitCost | Cost for a single unit of this item. (Set by the catalog). |
| **decimal** UnitList | Suggested price for a single unit of this item. (Set by the catalog). |

## IPriceCheck

An IPriceCheck object is used to hold all of the information necessary for performing a price check on the catalog. It holds information necessary to build the request (usually via an XML service) and also store the response.

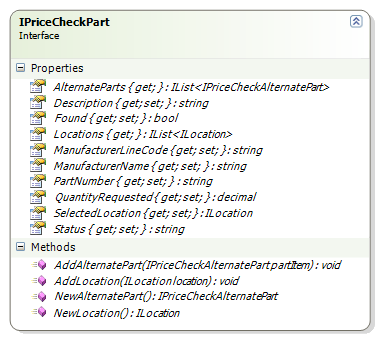
******

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** DeliveryOption | Delivery Option to consider for price check. (This property is currently ignored by Manager SE and is reserved for future use.) |
| **IList<IPriceCheckPart>** Parts | The list of parts to use when doing a price check |

## IPriceCheckPart

The IPriceCheckPart represents an individual part that is used for performing a price check. A list of IPriceCheckPart items is contained within an IPriceCheck object.

Information about the unit cost, unit list, and quantity available are contained within the ILocation object. The SelectedLocation is the original location, and it must be contained in the list of Locations. The others Locations are alternate locations. Even if a catalog does not support alternate locations, one location entry must be created to hold the pricing and availability of the part.

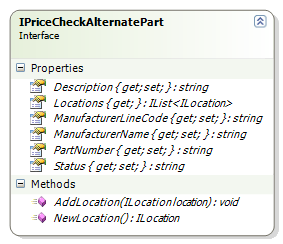
******

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **IList<IPriceCheckAlternatePart>** AlternateParts | List of alternate parts that can be substituted for this part |
| **string** Description | A description for the part. (This property should not be updated unless it is passed in as a blank string) |
| **bool** Found | Indicates whether the catalog recognizes this part. (Set by the catalog) |
| **IList<ILocation>** Locations | A list of locations where the part may be ordered from |
| **string** ManufacturerLineCode | Manufacturer Line Code of part. (Set by the host application) |
| **string** ManufacturerName | The manufacturer name for the part |
| **string** PartNumber | Part Number of part (Set by the host application) |
| **decimal** QuantityRequested | Requested Quantity of this part. (set by the host application) |
| **ILocation** SelectedLocation | The selected location for the part. |
| **string** Status | Custom Status of a processed part. Reserved for future use. |

|  |  |
| --- | --- |
| METHOD | MEANING |
| AddAlternatePart(**IPriceCheckAlternatePart** partItem) | Adds a new alternate part to **AlternateParts**. |
| AddLocation(**ILocation** location) | Adds a new location to Locations. |
| **IPriceCheckAlternatePart** NewAlternatePart() | Creates a new alternate part to be added to **AlternateParts** |
| **ILocation** NewLocation() | Creates a new ILocation that can be added to **Locations** |

## IPriceCheckAlternatePart

IPriceCheckAlternatePart represents an alternate part that can be substituted for an original part.

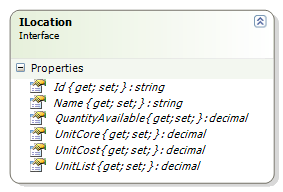


|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Description | Description for the alternate part |
| **IList<ILocation>** Locations | List of locations where the alternate part may be ordered from |
| **string** ManufacturerLineCode | Manufacturer line code of alternate part. |
| **string** ManufacturerName | Manufacturer name of the alternate part |
| **string** PartNumber | Part Number of the alternate part |
| **decimal** QuantityRequested | Requested Quantity of this part. (set by the host application) |
| **string** Status | Custom Status of a processed part. Reserved for future use. |

|  |  |
| --- | --- |
| METHOD | MEANING |
| AddLocation(**ILocation** location) | Adds a new location to the alternate part. |
| **ILocation** NewLocation() | Creates a new ILocation that can be added to this alternate part. |

## ILocation

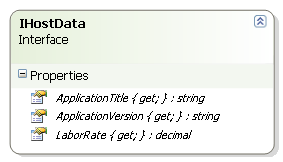
An ILocation object references a location where a part may be ordered from. It is used in an IPriceCheckPart to indicate the location of the original part along with the pricing and availability.

******

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Id | Id of the location (as determined by the catalog) (Max length 8 characters) |
| **string** Name | Name of the location (Max length 20 characters) |
| **decimal** QuantityAvailable | Quantity of this part item available at this location. |
| **decimal** UnitCore | Core charge for this item |
| **decimal** UnitCost | Cost for a single unit of this item |
| **decimal** UnitList | Suggested price for a single unit of this item |

## IHostData

IHostData represents information that the host application makes available to catalogs. An object supporting this interface is passed to both **GetCatalog** and **VendorSetup**.

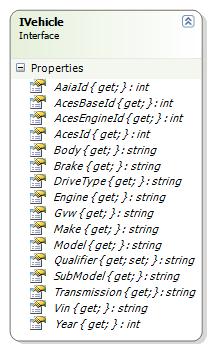


|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** ApplicationTitle | Title of the host application. |
| **string** ApplicationVersion | The first 3 components of the current version of the host application. |
| **decimal** LaborRate | The default hourly rate the host application uses to calculate labor items |

## IVehicle

IVehicle represents the user selected vehicle for use in filtering catalog part searches. The vehicle is defined in the host application and then passed to the catalog on a call to **GetCatalog**. Not all properties in the IVehicle object are guaranteed to be defined. Catalogs should attempt to identify the vehicle based on available information.

The **Qualifier** may be modified by the catalog. All other properties are read-only.



|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **int** AaiaId | AAIA (Automotive Aftermarket Industry Association) identifier for this vehicle. |
| **int** AcesBaseId | ACES (Aftermarket Catalog Enhanced Standard) Base identifier for this vehicle |
| **int** AcesEngineId | ACES Engine identifier for this vehicle |
| **int** AcesId | ACES identifier for this vehicle. |
| **string** Body | Body style of this vehicle (like: "sedan" or "wagon") |
| **string** Brake | Description of brakes installed on this vehicle |
| **string** DriveType | Drive train type for this vehicle |
| **string** Engine | Description of engine in this vehicle |
| **string** Gvw | GVW (Gross Vehicle Weight) of this vehicle |
| **string** Make | Make of the vehicle |
| **string** Model | Model of the vehicle |
| **string** Qualifier | String that can be used by the catalog to store vehicle information. The host application will store this information and pass it back to the catalog each time the **GetCatalog** method is called. |
| **string** SubModel | Submodel of the vehicle |
| **string** Transmission | The type of transmission installed on this vehicle. |
| **string** Vin | VIN (Vehicle Identification Number) of the vehicle |
| **int** Year | Model year of the vehicle |

## IVehicleAcesProvider

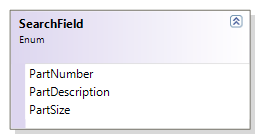
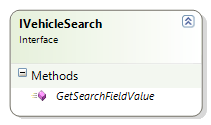
IVehicleAcesProvider represents all AcesID item. The single method “GetAcesId(AcesId id)”, takes in a single parameter of AcesId from the Enum “AcesId” and returns the value for the selected Vehicle’s AcesID. This Interface is easily extendable using the Enum “AcesId”. The method will return zero if this ID is not selected completely in the Repair Order.

## 

|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **int** BaseVehicleID | ACES (Aftermarket Catalog Enhanced Standard) Base Vehicle identifier for this vehicles Year Make and Model. |
| **int** VehicleID | ACES specific vehicle identifier for this vehicle. |
| **int** EngineBaseID | ACES identifier for this vehicles base engine. |
| **int** EngineConfigID | ACES identifier for this vehicles specific engine. |
| **int** SubModelID | ACES identifier for this vehicles sub model. |
|  |  |

## IVehicleSearch

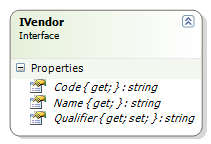
IVehicleSearch is similar to IVehicleAcesProvider (an additional interface IVehicle might have implemented). This additional interface provides the catalog with search criteria a user may have entered into a search box before entering into a catalog’s GoShopping session. For example, the tire promotional screen has the size field that, if text is present, will pass as SearchField.PartSize with the text value. This allows a catalog, for example, to prefill a size combo box with the requested size. If a given value is not provided, the returned value from GetSearchFieldValue will be String.Empty.



|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **PartNumber** | Any part number SE may have passed in (currently, not used). |
| **PartDescription** | Any description SE may have passed in (currently, not used). |
| **PartSize** | Any part size value SE may have passed in - meaning/format of text is up to the catalog. |
|  |  |

## IVendor

IVendor represents a parts Vendor. The Code and Name are configured in the host application. The Qualifier represents custom information as defined by the catalog (typically username and password coded as XML).



|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **string** Code | A short code which uniquely identifies the vendor |
| **string** Name | Name of the vendor |
| **string** Qualifier | String that can be used by the catalog to store custom vendor information. **Important:** This value can only be changed and persisted when returned from VendorSetup. Any other time, changes to this field will be discarded. |

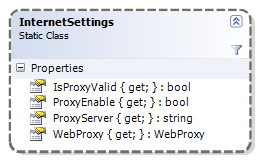
# Common Helper Classes

Included with the SDK are a collection of helper classes which will assist you in developing your parts catalog. These include a class for setting up an Internet proxy for web requests, creating log entries in a standard text-based log file, and various Utility methods.



## InternetSettings

InternetSettings provides helper methods to use Windows Internet Settings from the registry. This allows you to get a WebProxy object configured with the local machine’s Internet proxy settings.



|  |  |
| --- | --- |
| PROPERTY | MEANING |
| **bool** IsProxyValid | Does the local machine have an Internet proxy address configured and enabled in the Windows Internet Settings? |
| **bool** ProxyEnable | Does the local machine has a proxy enabled in the Windows Internet Settings |
| **string** ProxyServer | Retrieves the proxy server address as configured in the Windows Internet Settings |
| **WebProxy** WebProxy | Retrieves a WebProxy object configured with the proxy defined in the local Windows Internet Settings if a valid proxy is configured and enabled, otherwise return null. |

## Note: Calls to Web Services should use the following code…

if (InternetSettings.IsProxyValid)

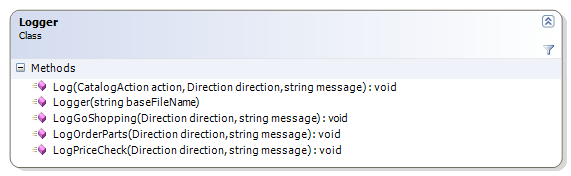
{

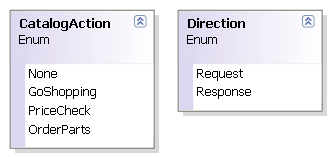
service.Proxy = InternetSettings.WebProxy;

}

## where the Proxy property of the service object implements the System.Net.IWebProxy interface. Examples of such classes are HttpWebRequest or SoapHttpClientProtocol Logger

Logger provides a set of methods to help you log catalog activity to a standard text file. Logging should be used to help customer support diagnose issues they experience when using a parts catalog.



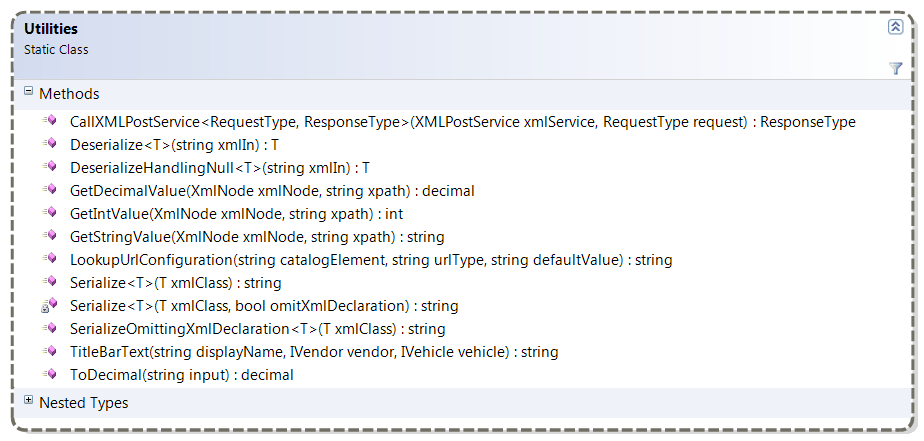


|  |  |
| --- | --- |
| CONSTRUCTOR | MEANING |
| Logger(**string** baseFileName) | Creates a logger object for writing log entries to a text file |

|  |  |
| --- | --- |
| METHOD | MEANING |
| **void** Log(**CatalogAction** action, **Direction** direction, **string** message) | Write a log entry to the text file: baseFileName.{**CatalogAction**}.Log |
| **void** LogGoShopping(**Direction** direction, **string** message) | Write a log entry to the text file: baseFileName.GoShopping.Log |
| **void** LogOrderParts(**Direction** direction, **string** message) | Write a log entry to the text file: baseFileName.OrderParts.Log |
| **void** LogPriceCheck(**Direction** direction, **string** message) | Write a log entry to the text file: baseFileName.PriceCheck.Log |

## Utilities

The Utilities class contains static helper methods which help with serialization of objects to XML and deserialization of objects from XML.



|  |
| --- |
| METHOD |
| **ResponseType** CallXMLPostService**<RequestType, ResponseType>(XMLPostService** xmlService**, RequestType** request**)**  Helper method to serialize a class, send it to a XMLPostService, and deserialize the result into a class. |
| **T** Deserialize<**T**>(**string** xmlIn)  Deserialize a XML string to a type **T**. |
| **T** DeserializeHandlingNull<**T**>(**string** xmlIn) **where** **T** : **new**()  Deserializes a XML string to a type **T**. if the string is null or empty, then a default object of type **T** is returned. |
| **string** LookupUrlConfiguration(**string** catalogElement, **string** urltype, **string** defaultValue)  Lookup a catalog URL configuration from the Catalogs.config XML file which can be created in the Catalogs folder. This would be used to use a non-production server for catalog testing. |
| **string** Serialize<T>(**T** xmlClass)  Transforms a class into an XML string. |
| **string** SerializeOmittingXmlDeclaration<T>(**T** xmlClass)  Transforms a class into an XML string without the XML declaration |
| **string** TitleBarText**(string** displayName**, IVendor** vendor**, IVehicle** vehicle**)**  Constructs the string which may be displayed in the parts catalog title bar |

## Catalog Implementation Help

The items which are added to the cart maintain the order in which they were inserted. This means if you want to place a note before a labor or part item, simply add the note just before you add the cart item you want to annotate.

### Sample Code

Fig. 1.1 – Creating a new part item from an ICart object and adding it to a shopping cart

// create a new part item and add it to a shopping cart

**public void** Fill(ICart shoppingCart)

{

IPartItem part = shoppingCart.NewPartItem();

part.PartNumber = partNumber;

part.ManufacturerLineCode = mfgCode;

part.ManufacturerName = mfgName;

part.Description = description;

part.UnitList = listPrice;

part.UnitCost = cost;

part.UnitCore = corePrice;

part.Quantity = qtyRequested;

shoppingCart.Add(part);

}

|  |
| --- |
| // create a new part item and add it to a shopping cart  **public void** Fill(ICart shoppingCart)  {  IPartItem2 part = (IPartItem2)shoppingCart.NewPartItem();  part.PartNumber = partNumber;  part.ManufacturerLineCode = mfgCode;  part.ManufacturerName = mfgName;  part.Description = description;  part.UnitList = listPrice;  part.UnitCost = cost;  part.UnitCore = corePrice;  part.Quantity = qtyRequested;  part.IsTire = isTire;  part.Size = size;  shoppingCart.Add(part);  } |

Fig 1.2 – Creating a new part item (version 2) from an ICart and adding it to a shopping cart

|  |
| --- |
| public bool GoShopping(ICart cart)  {  var cartOrder = cart as ICartOrder;  if (cart != null)  {  // ICartOrder is supported by Managaer and “Place Order” can be supported in the catalog. When the user wishes to close & save the order, use this interface to add ordered parts. Continue to use ICart for non-ordered parts/items.  }  else  {  // Manager does not support “Place Order” from catalog, use only ICart interface for parts/items.  }  } |

Fig 1.3 – Demonstrates how to check for ICartOrder argument in GoShopping for adding place order function to catalog.