Enhancement Guide
SAP Direct Store Delivery
Document Version: 1.0 – 2017-09-15

CUSTOMER

SAP Direct Store Delivery

Enhancement Guide for Windows Mobile

Typographic Conventions

Type Style	Description
Example	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Textual cross-references to other documents.
Example	Emphasized words or expressions.
EXAMPLE	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE	Keys on the keyboard, for example, F2 or ENTER.

Document History

Version	Date	Change
1.0	2017-09-15	

Table of Contents

1 Introduction

1.1 About this Document

This document is intended for a target audience of application programmers and administrators.

The following are described in this document:

- The *Extensibility Plugin*

  This document aims to enable you to understand what the *Extensibility Plugin* means and to show you how to implement modules using this plugin.

  The *Extensibility Plugin* is a way to extend the functionality of the *SAP Direct Store Delivery* mobile app.

  In this document, you get a description of how to build modules to extend an existing dialog and how to replace one dialog with another.

  The *SAP Direct Store Delivery* mobile app is built in C# and uses the *Model View Controller* (MVC) paradigm. The controller is the most important element and is responsible for getting dialogs started and closed and for ensuring that the users' input is handled correctly. Every dialog consists of exactly one controller. Furthermore, there is one controller, called the main controller, which basically controls all others.

- Pricing requirements and formula routines
  This document describes how customers can implement their own requirements and formula routine to customize the determination of condition records and the evaluation of conditions in pricing.
- Mobile reports
  This document describes how customers can edit the report templates or create new ones for the SAP Direct Store Delivery front end component.

1.2 Terminology

Term	Explanation
API	application programming interface
DAL	data access layer
DLL	dynamic link library
JSON	JavaScript Object Notation.
.Net CF 3.5 SP1	Microsoft .NET Compact Framework 3.5 Service Pack 1
VS 2008 SP1	Microsoft Visual Studio 2008 Service Pack 1
MVC	Model View Controller
MVVM	Model View ViewModel
plugin	A plugin is a software component that adds a specific feature to an existing software application.
pricing engine	A mobile app for performing price calculations in offline mode. The pricing engine is connected to *SAP Direct Store Delivery* on the mobile device where it uses the condition technique to determine the prices and taxes of products in sales documents.
SAP Direct Store Delivery	An SAP mobile app that supports the process of selling and distribution of goods directly to the customer store bypassing the retailer warehouses.

2 SAP Direct Store Delivery .Net API

2.1 Build Process

Both Microsoft Visual Studio 2008 Service Pack 1 (VS 2008 SP1) and Microsoft .Net Compact Framework 3.5 SP1 (.Net CF 3.5 SP1) are required since the development targets Windows Mobile Classic 6.x Professional. All the required APIs are available in C# and assemblies. In order to set up your project, you have to create a new C# DLL project in VS 2008 SP1.

Once you have created the project, you have to add references to the assemblies, for example, the following:

- SAPCD.MobileClient.Framework.dll
- SAPCD.MobileClient.Dialog.dll
- SAPCD.MobileClient.Dialog.WMForms.dll
- SAPCD.MobileClient.ExtensibilityPlugin.dll

> **Note**
>
> This list is not final. You may be required to add further assemblies.

In your C# DLL project, you have to create a class that implements the *IExtensibilityPlugin* interface that identifies the resulting DLL as a DLL with extensions. Since this extension is under your control, you can decide which configuration capabilities you want to use and how to read them.

The interface itself that follows is quite slim:

```
namespace com.sapcd.mobileclient.extensibilityplugin.Interfaces
{
	/// <summary>
	/// The extensibility plug in interface.
	/// </summary>
	/// <author>SAP CD</author>	public interface IExtensibilityPlugin
	{
		/// <summary>
		/// Gets the ID of this plugin as string.
		/// </summary>
		/// <value>The ID of this plugin as string.</value>
		string PluginId
		{
			get;
		}

		/// <summary>
		/// Probe for a possible controller extension.
		/// </summary>
```

```
            /// <param name="controllerType">The controller type to look up.</param>
            /// <returns>The input argument, if the controller should not be
replaced. Otherwise the replaced controller type.</returns>
            Type GetControllerExtended(Type controllerType);


      }
}
```

Since the architecture relies heavily on the MVC / MVVM approach, all entry points are done via controller extension / derivation or replacement. In *GetControllerExtended*, you decide what to do and what to use both ways (back and forth).

2.2 Example of an Extension Configuration Using JSON

If you build an extension DLL, you are free to use any kind of configuration you want. For the purposes of this document, a JSON based configuration is used as an example.

You can freely select the structure. For the purposes of this example, it has the following structure:

```
[
	{
		"SourceDllFile": "SAPCD.MobileSalesClient.Application.dll",
		 "Source":        "MseMainDialogController",
		 "TargetDllFile": "MobileSalesClientExtensibilityTest.dll",
		 "Target":        "MseMainDialogExtendedController",
		 "Replace":       true
	},
	{
		"SourceDllFile": "SAPCD.MobileSalesClient.Application.dll",
		 "Source":        "AboutViewController",
		 "TargetDllFile": "MobileSalesClientExtensibilityTest.dll",
		 "Target":        "AboutViewExtendedController",
		 "Replace":       false
	},
	{
		"SourceDllFile": "SAPCD.MobileSalesClient.Application.dll",
		 "Source":        "ReportPreviewController",
		 "TargetDllFile": "MobileSalesClientExtensibilityTest.dll",
		 "Target":        "PreReportsController",
		 "Replace":       false
	}
```

The JSON file consists of the following three entries and is structured like every standard JSON file:

```
Every entry consists of five parameters :


“SourceDllFile”     describes the module (DLL) containing the source controller
“Source”           describes the class of the source controller
                  (the controller type)
“TargetDLLFile”     describes the module (DLL) containing the target controller
“Target”            describes the class of the target controller
                  (the controller type)
“Replace”          tells the plugin whether to exchange the source against the
                  target controller type (bool)
```

The *Replace* parameter is useful for allowing testers to exchange controller types even if they do not know the class names (controller types) and the file names of the modules containing them. In this way, a tester can run this by just changing the *Replace-Entry* from false to true and vice versa, where true means to exchange the controller types.

> **Note**
>
> Because the *Replace*-parameter is converted to a boolean-data type, only true and false are valid.

2.3 Examples

2.3.1 Example 1 - Controller Replacement

In this section, the complete replacement of a controller with another one is described.
If you decide to change the behavior of the application by implementing a new dialog, one way to do this is to replace one dialog with another one.
In this case, you do not inherit from another (already existing) dialog but instead you implement everything from scratch.

> **Note**
>
> As described earlier in this document, the application is based on the MVC paradigm.

Each dialog consists of the following four parts (C# modules):

- The controller class (PreReportsController)
- The view class (PreReportsView)
- The data view class (PreReportsDataView)
- The model class (PreReportsModel)

As a simple example in this document, the *Report-Preview* dialog is replaced with a new one. The UI of this dialog is the same as the original but with the implementation of a menu and a call to another dialog (*About* dialog), which shows the device and user information.
Therefore, in this example, the difference between the dialogs is just a menu and the menu handling.

To get the exchange to work, you create a JSON-file as follows:

```
[
	{
		"SourceDllFile": "SAPCD.MobileSalesClient.Application.dll",
		"Source":        "ReportPreviewController",
		"TargetDllFile": "MobileSalesClientExtensibilityTest.dll",
		"Target":        "PreReportsController",
		"Replace":       true
	}
]
```

Every time the *ReportPreviewController* is activated, the *ExtensibillityPlugin* exchanges the source *controllertype* with the *PreReportsController* type.

2.3.1.1 The Controller Class

The controller class inherits from `MobileDialogController,` overrides some methods, and implements some properties.

```
/// <summary>
/// The items which represent the actions possible on the PreReports dialog
// </summary>
public enum ActionKeys
{
        None,
        About,
        Back,
        Font,
        FontSize6,
        FontSize8,
        FontSize10,
        FontSize12
}
public override List<VolatileDataContainerParameter> InputParameterList
{
        get{}
}
/// <summary>
/// Gets the type of the view.
/// </summary>
/// <value>
/// The type of the view.
/// </value>
public override Type ViewType
{
        get
        {
         return typeof(PreReportsView);
        }
}

/// <summary>
```

```
/// Gets the type of the model.
/// </summary>
/// <value>
/// The type of the model.
/// </value>
public override Type ModelType
{
	get
	{
		return typeof(PreReportsModel);
	}
}

/// <summary>
/// Gets a value indicating whether this instance can be closed.
/// </summary>
/// <value>
/// <c>true</c> if this instance can be closed; otherwise, <c>false</c>.
/// </value>
public override bool CanCloseController
{
	get
	{
		return true;
	}
}

/// <summary>
/// Gets the type of the action enum.
/// </summary>
/// <value>
/// The type of the action enum.
/// </value>
public override List<Type> ActionEnumType
{
	get
	{
		List<Type> aetList = new List<Type>() { typeof(ActionKeys) };
		aetList.AddRange(base.ActionEnumType);
		return aetList;
	}
```

```
}

/// <summary>
/// Gets the typed view.
/// </summary>
private PreReportsView TypedView
{
        get
        {
                return this.typedView ?? (this.typedView = (PreReportsView)View);
        }
}

/// <summary>
/// Gets the specific model.
/// </summary>
private PreReportsModel TypedModel
{
        get
        {
                return this.typedModel ?? (this.typedModel = (PreReportsModel)Model);
        }
}

/// <summary>
/// Gets the typed data view.
/// </summary>
private PreReportsDataView TypedDataView
{
        get
        {
                return this.typedDataView ?? (this.typedDataView =
                (PreReportsDataView)TypedModel.DataView);
        }
}
/// <summary>
/// Registers the action. That means the action will be activated if the user
/// raises an event. i.e. clicks on a button or menuitem
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
```

```
protected override void RegisterAction(Enum key, IActionGroup ag)
{
	ActionKeys tmpKey = (ActionKeys)key;
	switch (tmpKey)
	{
		case ActionKeys.Font:
		{
			//Nothing to do here
			break;
		}
		case ActionKeys.About:
		{
			ag.Action += AboutActionHandler;
			break;
		}
		case ActionKeys.Back:
		{
			ag.Action += BackActionHandler;
			break;
		}
		case ActionKeys.FontSize6:
		case ActionKeys.FontSize8:
		case ActionKeys.FontSize10:
		case ActionKeys.FontSize12:
		{
			ag.Action += FontSizteActionHandler;
			break;
		}
	}

	base.RegisterAction(key, ag);
}

/// <summary>
/// Deregisters the action.
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
protected override void DeregisterAction(Enum key, IActionGroup ag)
{
	ActionKeys tmpKey = (ActionKeys)key;
```

```
        switch (tmpKey)
        {
                case ActionKeys.Font:
                {
                        //Nothing to do here
                        break;
                }
                case ActionKeys.About:
                {
                        ag.Action -= AboutActionHandler;
                        break;
                }
                case ActionKeys.Back:
                {
                        ag.Action -= BackActionHandler;
                        break;
                }
                case ActionKeys.FontSize6:
                case ActionKeys.FontSize8:
                case ActionKeys.FontSize10:
                case ActionKeys.FontSize12:
                {
                        ag.Action -= FontSizteActionHandler;
                        break;
                }
        }

        base.DeregisterAction(key, ag);
}
#region Action Handlers


/// <summary>
/// Fontsize action handler. Shows the text in a newly selectd fontsize.
/// </summary>
/// <param name="sender">The sender.</param>
/// <param name="args">The <see
/// ref="com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroupEventArgs"/>
/// instance containing the event data.</param>
private void FontSizteActionHandler(IActionGroup sender, IActionGroupEventArgs args)
```

```
        float size = 0;
        MenuItem mi = (MenuItem)args.Sender;
        if (mi.Text.StartsWith("Fontsize "))
        {
                string[] arr = mi.Text.Split(' ');
                if (arr.Length == 2)
                {
                        size = (float)Convert.ToDecimal(arr[1]);
                        TypedView.txtMessages.Font = new Font("Courier New", size,
                        FontStyle.Regular);
                }
        }
}

/// <summary>
/// About action handler.
/// </summary>
/// <param name="sender">The sender.</param>
/// <param name="args">The <see
/// cref="com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroupEventArgs"/>
/// instance containing the event data.</param>
private void AboutActionHandler(IActionGroup sender, IActionGroupEventArgs args)
{
        var ctrl = asterDialogController.CreateController(typeof(AboutViewController));
                MasterDialogController.PushController(ctrl);
}

/// <summary>
/// Backs the action handler.
/// </summary>
/// <param name="sender">The sender.</param>
/// <param name="args">The <see
/// cref="com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroupEventArgs"/>
/// instance containing the event data.</param>
private void BackActionHandler(IActionGroup sender, IActionGroupEventArgs args)
{
        MasterDialogController.PopController();
}
#endregion // Action Handlers
```

The *RegisterAction* and *DeregisterAction* methods are the important methods of the controller because they are responsible for getting the actions worked. They declare the *Actionhandler* methods to be called when an event arises to be handled by one of the *Actionhandler* methods. As soon as the controller gets stopped, it is responsible for getting the A*ctionhandlers* deregistered so that they cannot be called incorrectly.
The place where the controls, for example, buttons or menu items, are defined and bound to the action groups is in the view.

2.3.1.2 The View Class

The view class is responsible for building the dialog with the controls available in Windows Forms (here in our example).
The view builds the user interface.
The view class inherits from the `WMFormsDialogUserControl` class and consists of two partial class-parts, as does every class which is used in Windows Forms to build a dialog.
The dialog itself can be designed anyway you want. However, to get the look and feel of the whole application, it is recommended to do as follows (this takes place in the designer-part of the view-class):

```
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        this.headerBar = new com.sapcd.mobileclient.dialog.WMForms.Controls.MobileHeaderBar();
        this.btnBack = new om.sapcd.mobileclient.dialog.WMForms.Controls.MobileHeaderBarButton();
        this.pnlSyncView = new com.sapcd.mobileclient.dialog.WMForms.Controls.MobilePanel();
        this.txtMessages = new com.sapcd.mobileclient.dialog.WMForms.Controls.MobileTextBox();
        this.headerBar.SuspendLayout();
        this.pnlSyncView.SuspendLayout();
        this.SuspendLayout();
        //
        // headerBar
        //
        this.headerBar.Controls.Add(this.btnBack);
        this.headerBar.Location = new System.Drawing.Point(0, 0);
        this.headerBar.Name = "headerBar";
        this.headerBar.Size = new System.Drawing.Size(480, 58);
        //
        // btnBack
        //
        this.btnBack.Location = new System.Drawing.Point(3, 0);
        this.btnBack.Name = "btnBack";
        this.btnBack.Size = new System.Drawing.Size(110, 58);
        this.btnBack.TabIndex = 0;
        this.btnBack.Text = "btnBack";
        //
        // pnlSyncView
        //
        this.pnlSyncView.BackColor = System.Drawing.SystemColors.Info;
```

```
        this.pnlSyncView.Controls.Add(this.txtMessages);
        this.pnlSyncView.Location = new System.Drawing.Point(0, 58);
        this.pnlSyncView.Name = "pnlSyncView";
        this.pnlSyncView.Size = new System.Drawing.Size(480, 478);
        //
        // txtMessages
        //
        this.txtMessages.Dock = System.Windows.Forms.DockStyle.Fill;
        this.txtMessages.Location = new System.Drawing.Point(0, 0);
        this.txtMessages.Multiline = true;
        this.txtMessages.Name = "This is a new controller...";
        this.txtMessages.ReadOnly = true;
        this.txtMessages.Size = new System.Drawing.Size(480, 478);
        this.txtMessages.TabIndex = 0;
        //
        // PreReportsView
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(192F, 192F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Dpi;
        this.Controls.Add(this.pnlSyncView);
        this.Controls.Add(this.headerBar);
        this.Name = "PreReportsView";
        this.Size = new System.Drawing.Size(480, 536);
        this.Controls.SetChildIndex(this.headerBar, 0);
        this.Controls.SetChildIndex(this.pnlSyncView, 0);
        this.headerBar.ResumeLayout(false);
        this.pnlSyncView.ResumeLayout(false);
        this.ResumeLayout(false);

}
```

The important elements are the header bar, the back-button, and the panel, which is the container for the textbox. The menu items are built in the second part of the view class.

In this example, where you do not extend a controller but create a new one, you have to build a menu but not from scratch.

In the (`WMFormsDialogUserControl`) base class, a menu is already declared and contains two empty items. The first item represents the left one in the menu bar while the second one represents the right one.

To put in your own menu items, check that they are not already there and then put them in.

You can only put in two menu items, where each of them can obtain as many items as you want, as follows:

```
/// <summary>
/// Gets the dialog menu.
/// </summary>
protected override List<MenuItem> DialogMenu
{
	get
	{
		if (menuItems.Count == 0)
		{
			menuItems.Add(mnuFont);
			mnuFont.MenuItems.Add(mnuFont6);
			mnuFont.MenuItems.Add(mnuFont8);
			mnuFont.MenuItems.Add(mnuFont10);
			mnuFont.MenuItems.Add(mnuFont12);
			menuItems.Add(mnuAbout);
		}
		return menuItems;
	}
}
```

In this example, the left menu item is the *Font* menu with 4 further items. The right one only contains the *About* menu.

Another important thing is to override some methods as follows:

```
/// <summary>
/// Initializes this instance.
/// </summary>
public override void Initialize()
{
	Localize();
	this.txtMessages.Font = new Font("Courier New", 8, FontStyle.Regular);
	base.Initialize();
}


/// <summary>
/// Localizes this instance.
/// </summary>
public override void Localize()
{
```

```
        // window title
        Title = "Pre Reports";
        // buttons
        // Usually use localization mechanism here
        mnuFont.Text = "Font";
        mnuAbout.Text = "About";
        mnuFont6.Text = "Fontsize 6";
        mnuFont8.Text = "Fontsize 8";
        mnuFont10.Text = "Fontsize 10";
        mnuFont12.Text = "Fontsize 12";
        btnBack.Text = MobileClientGlobalLocalizer.Singleton.Global_btnBack.Translation;
        // others
        headerBar.LblTitle.Text = Title;
        base.Localize();
}
/// <summary>
/// Deregisters data binding.
/// </summary>
public override void DeregisterDataBinding()
{
        // Deregister all bindings that were previously registered.
        bindingSource = null;
        txtMessages.DataBindings.Clear();
        base.DeregisterDataBinding();
}

/// <summary>
/// Registers the data binding.
/// </summary>
public override void RegisterDataBinding()
{
        try
        {
                // Add bindings
                bindingSource = new BindingSource
                {
                        DataSource = this.dataView
                };

                txtMessages.DataBindings.Add("Text", bindingSource, "PreviewText", true,
                DataSourceUpdateMode.OnPropertyChanged);
```

```
        }
        catch (Exception e)
        {
                Logger.Singleton.Log(e);
        }
        base.RegisterDataBinding();
}

/// <summary>
/// Registers the action.
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
public override void RegisterAction(Enum key, IActionGroup ag)
{
        PreReportsController.ActionKeys tmpKey = (PreReportsController.ActionKeys)key;
        switch (tmpKey)
        {
                case PreReportsController.ActionKeys.Font:
                // ag.Add(this.mnuStart);
                break;

                case PreReportsController.ActionKeys.About:
                ag.Add(this.mnuAbout);
                break;

                case PreReportsController.ActionKeys.Back:
                ag.Add(this.btnBack);
                break;

                case PreReportsController.ActionKeys.FontSize6:
                ag.Add(this.mnuFont6);
                break;

                case PreReportsController.ActionKeys.FontSize8:
                ag.Add(this.mnuFont8);
                break;

                case PreReportsController.ActionKeys.FontSize10:
                ag.Add(this.mnuFont10);
                break;
```

```
                case PreReportsController.ActionKeys.FontSize12:
                ag.Add(this.mnuFont12);
                break;
        }

        base.RegisterAction(key, ag);
}


/// <summary>
/// Registers the action.
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
public override void DeregisterAction(Enum key, IActionGroup ag)
{
        PreReportsController.ActionKeys tmpKey = (PreReportsController.ActionKeys)key;
        switch (tmpKey)
        {
                case PreReportsController.ActionKeys.Font:
                // ag.Remove(this.mnuStart);
                break;

                case PreReportsController.ActionKeys.About:
                ag.Remove(mnuAbout);
                break;

                case PreReportsController.ActionKeys.Back:
                ag.Remove(btnBack);
                break;

                case PreReportsController.ActionKeys.FontSize6:
                ag.Remove(mnuFont6);
                break;

                case PreReportsController.ActionKeys.FontSize8:
                ag.Remove(mnuFont8);
                break;

                case PreReportsController.ActionKeys.FontSize10:
```

```
                ag.Remove(mnuFont10);
                break;

                case PreReportsController.ActionKeys.FontSize12:
                ag.Remove(mnuFont12);
                break;
        }

        base.RegisterAction(key, ag);
}

#endregion // IDialogView / Overrides

#region Methods

/// <summary>
/// Registers a data view. The dialog view should bind its UI controls to the
/// properties of the data view.
/// </summary>
/// <param name="dv">The data view.</param>
public override void RegisterDataView(IDataView dv)
{
        base.RegisterDataView(dv);

        // The data view needs to register itself with the view, so we can set it
        // as the bindings datasource.
        this.dataView = dv as PreReportsDataView;
}

#endregion // Methods
```

The two most important overridden methods are *RegisterAction* and *DeregisterAction*.
For each entry in the *ActionKey* enumeration property of the controller, there is a call to *RegisterAction* and *DeregisterAction*. In this method, the control, which causes an event when clicked, is registered so that the controller can call the appropriate event handler. This mechanism is already done by the framework and here you just have to register and deregister the controls to be observed.

2.3.1.3 The Data View Class

The data view class is the part that contains the data to be displayed and edited in the dialog.
The data view derives from the data view class.
Usually, the data view gets data via the `public override void SelectData(VolatileDataContainer selectionData)` method.
This method, which is described below, enables you to get data from a data container that is globally available in the application and pushed from one dialog to another. The data can be added or removed to or from the container or just read from it by using a key and/or a combination of a category and a key. The data items are treated as objects and have to be casted in the appropriate type.
Furthermore, in this method, the properties are filled with data from the container or from the database. That is, the properties are bound to the controls properties with data binding as described in the view class.

```
public class PreReportsDataView : DataView
{
        #region Member
        const string KEY_PRINT_REPORT = "PrintReport";
        const string KEY_PRINT_PRINTING_ENABLED = "PrintReportPrintingEnabled";
        const string KEY_ISCANCELCOPY = "IsCancelCopy";

        private string previewText;
        private string previewTitle;

        #endregion

        /// <summary>
        /// Selects the data.
        /// </summary>
        /// <param name="selectionData">The selection data.</param>
        public override void SelectData(VolatileDataContainer selectionData)
        {
                base.SelectData(selectionData);
                PrintReport = selectionData.GetObject<PrintReportBase>(KEY_PRINT_REPORT);
                PreviewText = PrintReport.Preview() ?? string.Empty;
                PreviewTitle = PrintReport.LocalizedHeader ?? string.Empty;
                IsPrintingEnabled = selectionData.GetObjectAsBool(KEY_PRINT_PRINTING_ENABLED,
                true);
        }

        /// <summary>
        /// Gets or sets the print report.
        /// </summary>
        /// <value>
```

```
/// The print report.
/// </value>
public PrintReportBase PrintReport
{
        get;
        set;
}

/// <summary>
/// Gets or sets a value indicating whether this instance is printing enabled.
/// </summary>
/// <value>
///     <c>true</c> if this instance is printing enabled; otherwise, <c>false</c>.
/// </value>
public bool IsPrintingEnabled
{
        get;
        set;
}

/// <summary>
/// Gets or sets the preview text.
/// </summary>
/// <value>
/// The preview text.
/// </value>
public string PreviewText
{
        get
        {
                return previewText;
        }
        set
        {
                if (previewText != value)
                {
                        previewText = value;
                        OnPropertyChanged("PreviewText");
                }
        }
}
```

```
        /// <summary>
        /// Gets or sets the preview title.
        /// </summary>
        /// <value>
        /// The preview title.
        /// </value>
        public string PreviewTitle
        {
                get
                {
                        return previewTitle;
                }
                set
                {
                        if (previewTitle != value)
                        {
                                previewTitle = value;
                                OnPropertyChanged("PreviewTitle");
                        }
                }

        }
```

The base class of the data view implements the *INotifyPropertyChanged* interface, which is also part of the data binding. As shown in the source code, every property bound to a control has to call the *OnPropertyChanged* method with the property name as a parameter. This makes the data binding work correctly.

Here, in this case, the calls to the following:

```
PrintReport = selectionData.GetObject<PrintReportBase>(KEY_PRINT_REPORT);
```

and

```
PreviewText = PrintReport.Preview() ?? string.Empty;
```

give us a *Print Report* object that gives us the reports text as a string to be shown in the *txtMessages* text box. This was already shown in the view where the data binding of the *txtMessages*-control took place. The *Text* property of the *txtMessages* is bound against the *PreviewText* property of the data view as follows:

```
txtMessages.DataBindings.Add("Text", bindingSource, "PreviewText", true,  ).
```

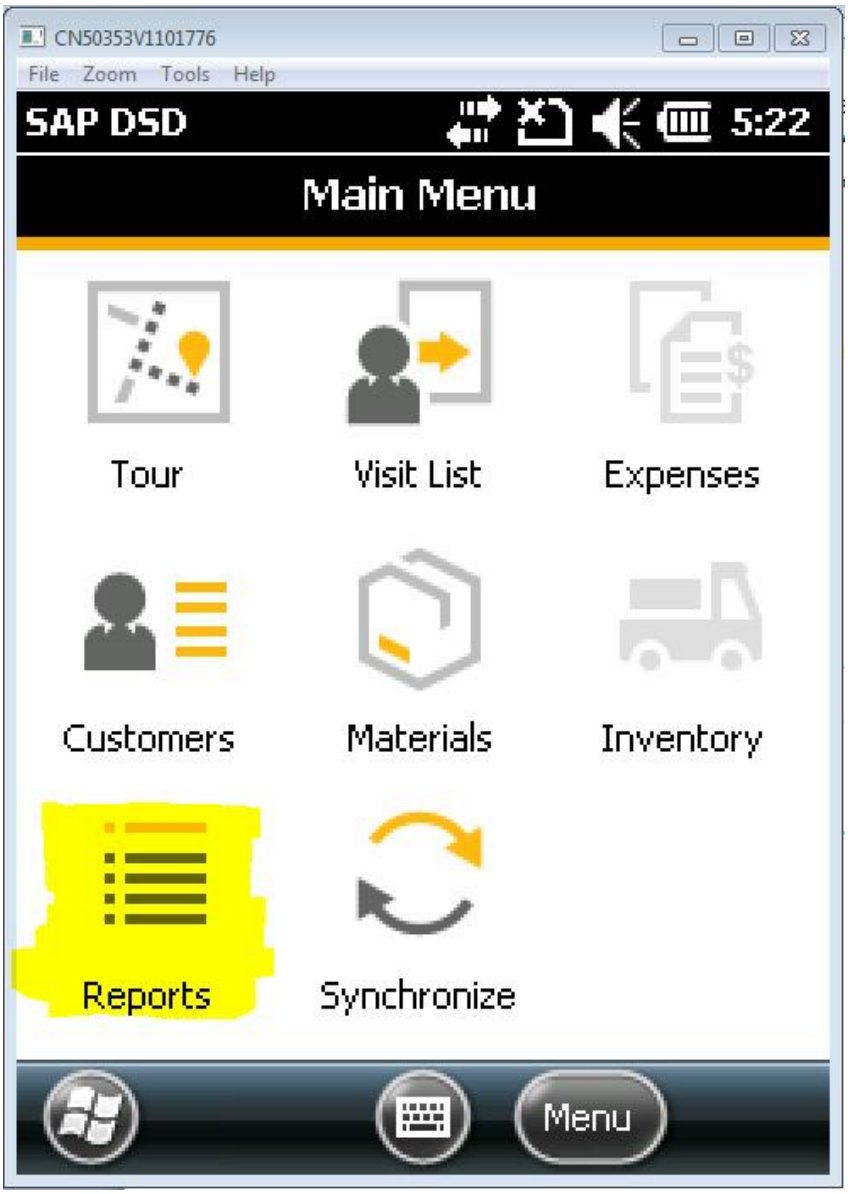
2.3.1.4 The Model Class

The model class derives from the *DialogModel* class and has only to override one property, that is, the *dataviewtype.* It has to be implemented as follows:

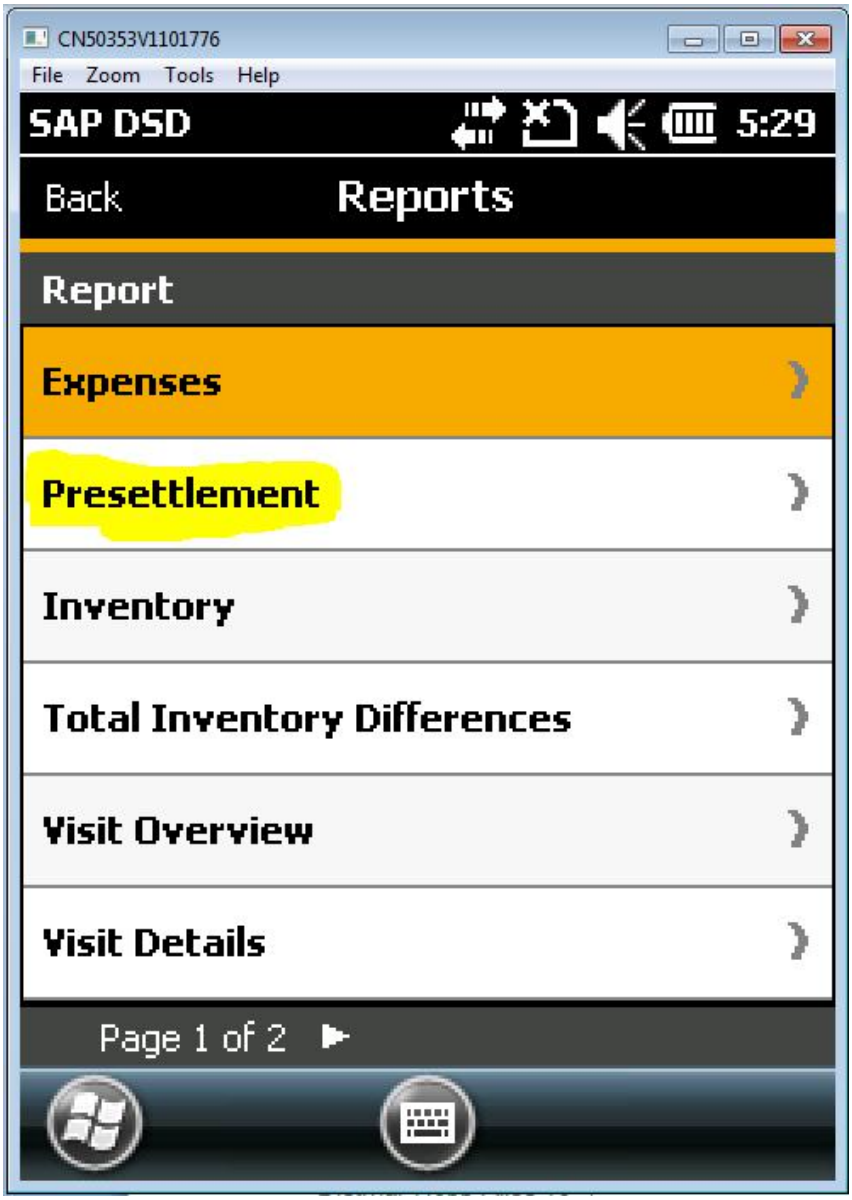
```
public class PreReportsModel : DialogModel
{
	#region Overrides

	/// <summary>
	/// Gets the type of the data view.
	/// </summary>
	/// <value>
	/// The type of the data view.
	/// </value>
	public override Type DataViewType
	{
		get
		{
			return typeof(PreReportsDataView);
		}
	}

	#endregion // Overrides
}
```

2.3.1.5 Screen Shots

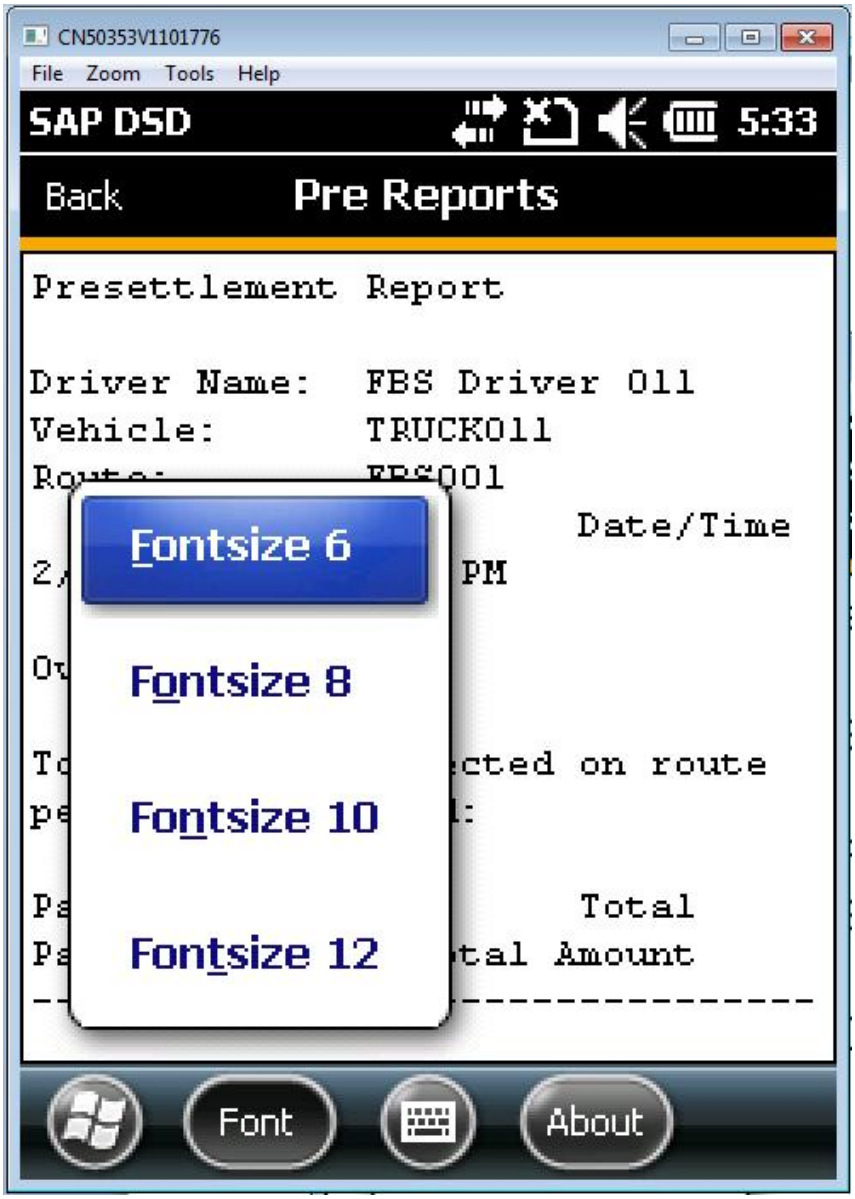
Main Menu



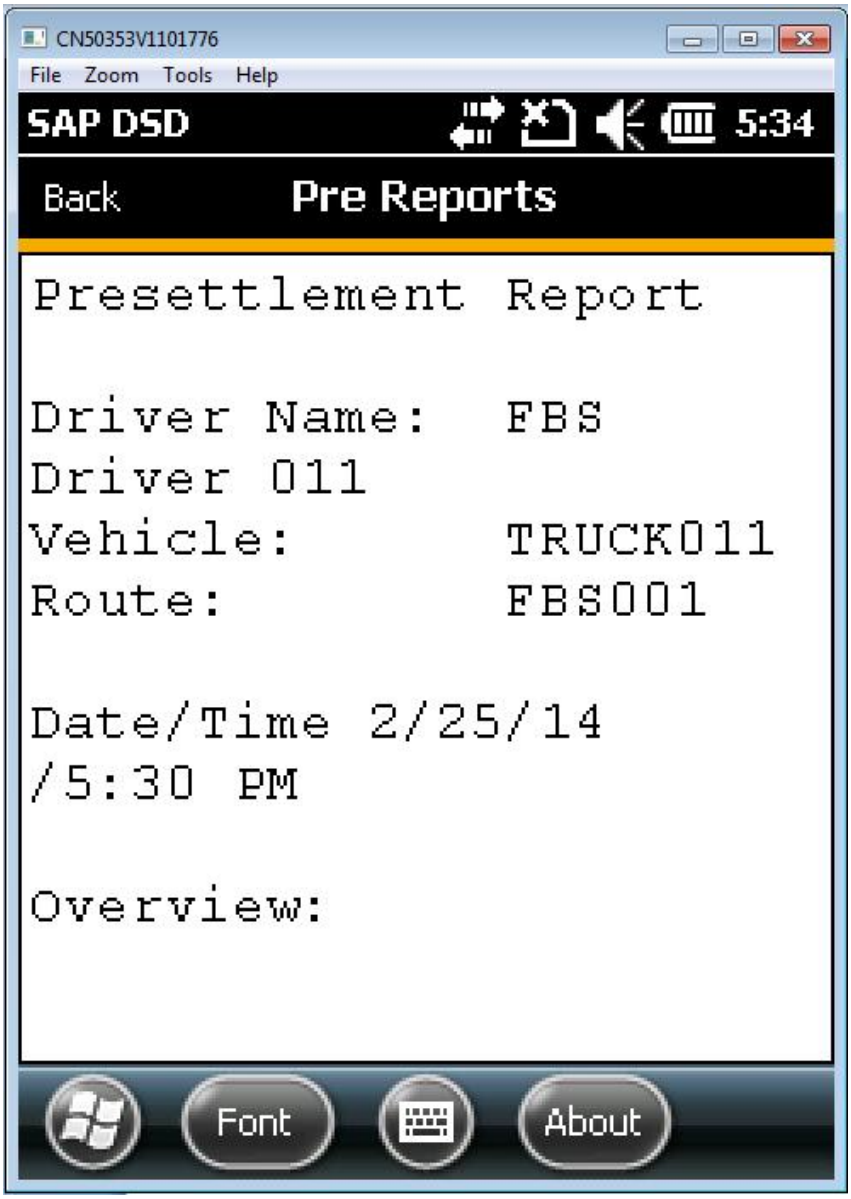
Reports



Pre Reports with Opened *Font* Menu



After choosing *Fontsize* **12**, the report shows the same data with a larger font as follows:



2.3.2 Example 2 - Controller Inheritance

This section describes inheriting from a controller to enlarge the existing controller's behavior.
This can be done by adding menu-items, methods, and/or properties in the appropriate modules like views, data views, and models.

The main module is the controller itself because it does all the work to get the actions registered, the view created and displayed, and so on.
The view is the part that is visible on the screen. In this example, you derive from the parent view and enlarge an existing menu.
To get a menu enlarged by one or more new menu-items, it is important to put the new menu-items into the existing menu of the inherited view, if one already exists.

First, let's take a look at the following view:

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using SAPCD.MobileSalesClient.Application.Main;
using com.sapcd.mobileclient.dialog.Interfaces.Common;
using com.sapcd.mobileclient.dialog.Interfaces.Model;
using com.sapcd.mobileclient.utilities.Common;

namespace MobileSalesClientExtensibilityTest.MseMainDialogExtended
{
        public class MseMainDialogExtendedView : MseMainDialogView
        {
                private MenuItem mnuAbout;

                #region Construction / Finalization

                /// <summary>
                /// Initializes a new instance of the <see cref="MseMainDialogExtendedView"/>
                /// class.
                /// </summary>
                public MseMainDialogExtendedView()
                        : base()
                {
                        mnuAbout = new MenuItem();
                }

                #endregion Construction / Finalization
```

```
#region Overrides

/// <summary>
/// Localizes this instance.
/// </summary>
public override void  Localize()
{
        base.Localize();
        mnuAbout.Text = "About";
}
/// <summary>
/// Registers the data view.
/// </summary>
/// <param name="dv">The dv.</param>
public override void RegisterDataView(IDataView dv)
{
        //this.dataView = (AboutViewDataView)dv;
        base.RegisterDataView(dv);
}

/// <summary>
/// Registers the action.
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
public override void RegisterAction(Enum key, IActionGroup ag)
{
        MseMainDialogExtendedController.ActionKeys tmpKey =
        (MseMainDialogExtendedController.ActionKeys)key;
        switch (tmpKey)
        {
                case MseMainDialogExtendedController.ActionKeys.AboutExt:
                ag.Add(this.mnuAbout);
                        break;
        }
        base.RegisterAction(key, ag);
}

/// <summary>
/// Registers the action.
```

```
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
public override void DeregisterAction(Enum key, IActionGroup ag)
{
        MseMainDialogExtendedController.ActionKeys tmpKey =
        (MseMainDialogExtendedController.ActionKeys)key;
        switch (tmpKey)
        {
                case MseMainDialogExtendedController.ActionKeys.AboutExt:
                ag.Remove(this.mnuAbout);
                break;
        }

        base.DeregisterAction(key, ag);
}


/// <summary>
/// Gets the menu items for the current dialog / view.
/// Only two items Send and Clear
/// </summary>
protected override List<MenuItem> DialogMenu
{
        get
        {
                int anz = 0;
                MenuItem mi = null;
                menuItems = base.DialogMenu;
                anz = this.menuItems.Count;
                if (anz > 0)
                {
                        mi = menuItems[anz - 1];
                }
                else
                {
                        mi = menuItems[0];
                }

                if (mi.MenuItems.Count != 0 && !mi.MenuItems.Contains(mnuAbout))
                {
```

```
                                mi.MenuItems.Add(mnuAbout);
                            }
                            return menuItems;
                        }
                    }

                    #endregion Overrides

                }
            }
```

In this case, you know that the parent already owns a menu so you can just extend the existing menu.

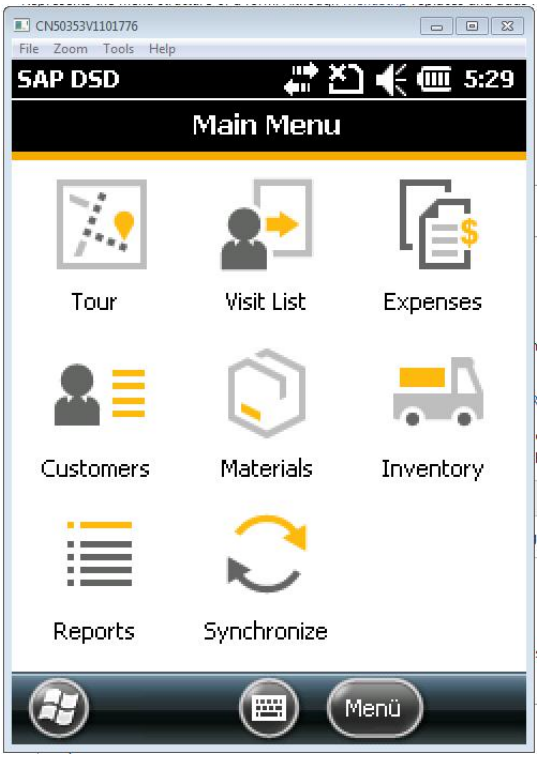
It is important to know that the menu contains two menu items where the first of them represents the left one while the second one represents the right one.
In this case, you added a new item (*mnuAbout*) to the right menu. The left one stays invisible.

Another way to add a menu is to change the default menu´s behavior.

In the class where the system-menu is built for the first time, the code results in a menu consisting of a left and a right menu-control. The left one is invisible and disabled and its *Text-Property* has an empty string as a value. The C# code is as follows:

```
/// <summary>
/// Gets the menu items for the current dialog / view.
/// </summary>
protected override List<MenuItem> DialogMenu
{
        get
        {
                if (this.menuItems.Count == 0)
                {
                        MenuItem mnuLeft = new MenuItem();
                        mnuLeft.Text = "";
                        mnuLeft.Enabled = false;
                        MenuItem mnuRight = new MenuItem();
                        mnuRight.Text = "Menu";

                        ParentWnd.BuildSystemMenu(mnuRight);

                        this.menuItems.Add(mnuLeft);
                        this.menuItems.Add(mnuRight);
                }
                return this.menuItems;
        }
}
```

This results in two menu-controls as follows with the left one being invisible:

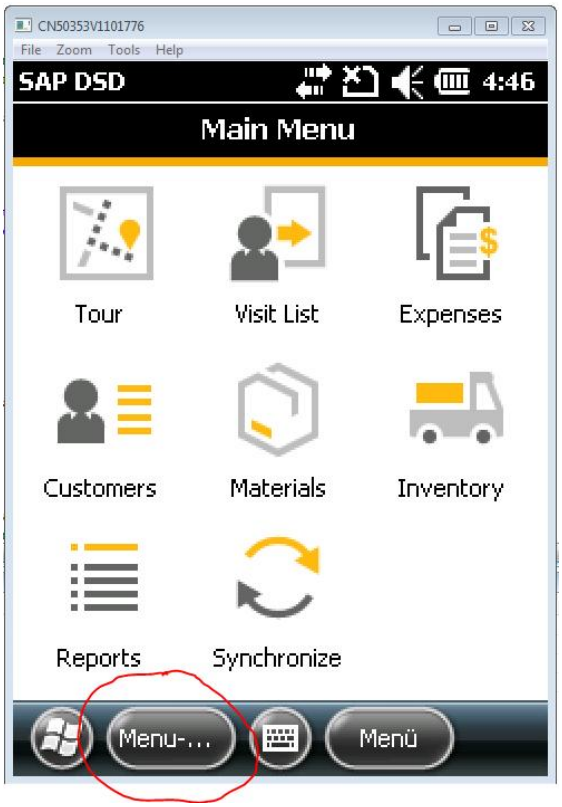


To get a menu to look like the following, you just have to change the default-menu´s behavior:

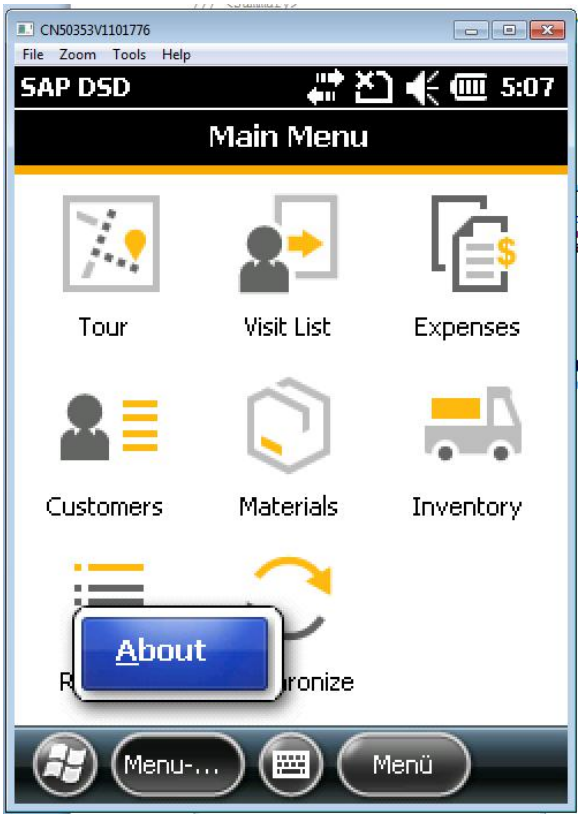
```
/// <summary>
/// Gets the menu items for the current dialog / view.
/// Only two items Send and Clear
/// </summary>
protected override List<MenuItem> DialogMenu
{
        get
        {
                int anz = 0;
                MenuItem mi = null;
                menuItems = base.DialogMenu;
                anz = menuItems.Count;
                if (anz > 0 && !menuItems[0].MenuItems.Contains(mnuAbout))
                {
                        menuItems[0].Enabled = true;
                        menuItems[0].Text = "Menu-Left";
                        menuItems[0].MenuItems.Add(mnuAbout);
                }
```

```
        return menuItems;
        }
}
```

In this case, you just enabled the left menu control, gave it another *Text* value, and added menu item to its *Menu Item* property. All the other code stays the same as in the example above. The result is as follows:



Choosing the *Menu-Control* looks as follows:



The menu works as described in the examples above.

To get submenus into a menu, you just have to add further menu items into the menu item´s *MenuItem* property as descibed in the following code snippet:

```
mnuDoThis = new MenuItem();
mnuDoThat = new MenuItem();
mnuDoSomething = new MenuItem();
mnuDoThis.Text = "DoThis";
mnuDoThat. ...

mnuAbout.MenuItems.Add(mnuDoThis);
mnuAbout.MenuItems.Add(mnuDoThat);
mnuAbout.MenuItems.Add(mnuDoSomething);

menuItems[0].Enabled = true;
menuItems[0].Text = "Menu-Left";
menuItems[0].MenuItems.Add(mnuAbout);
```

> **Note**
>
> Remember to register all *MenuItems* and *Actionkeys*.

2.3.2.1 Deriving a Controller

The controller derived from the parent controller looks as follows:

> **Note**
>
> The parent controller is the *MseMainDialogController*.

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using SAPCD.MobileSalesClient.Application.Main;
using SAPCD.MobileSalesClient.Application.UserViewPlugin.About;
using com.sapcd.mobileclient.dialog.Interfaces.Common;
using com.sapcd.mobileclient.utilities.Common;

namespace MobileSalesClientExtensibilityTest.MseMainDialogExtended
{
        class MseMainDialogExtendedController : MseMainDialogController
        {
                /// <summary>
                /// The Action keys
                /// </summary>
                public enum ActionKeys
                {
                        AboutExt
                }

                private MseMainDialogExtendedModel typedModel;
                private MseMainDialogExtendedView typedView;
                private MseMainDialogExtendedDataView typedDataView;

                public MseMainDialogExtendedController()
                   : base()
                {

                }

                /// <summary>
                /// Registers the action.
                /// </summary>
                /// <param name="key">The key.</param>
```

```
/// <param name="ag">The ag.</param>
protected override void RegisterAction(Enum key, IActionGroup ag)
{
        ActionKeys tmpKey = (ActionKeys)key;
        switch (tmpKey)
        {
                case ActionKeys.AboutExt:
                {
                        ag.Action += ExtAboutActionHandler;
                        break;
                }
        }
        base.RegisterAction(key, ag);
}


/// <summary>
/// Gets the type of the action enum.
/// </summary>
/// <value>
/// The type of the action enum.
/// </value>
public override List<Type> ActionEnumType
{
        get
        {
                List<Type> aetList = new List<Type>() { typeof(ActionKeys) };
                aetList.AddRange(base.ActionEnumType);
                return aetList;
        }
}

/// <summary>
/// De-Registers the action.
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
protected override void DeregisterAction(Enum key, IActionGroup ag)
{
        ActionKeys tmpKey = (ActionKeys)key;
        switch (tmpKey)
```

```
        {
                case ActionKeys.AboutExt:
                {
                        ag.Action -= ExtAboutActionHandler;
                        break;
                }
        }
        base.DeregisterAction(key, ag);
}

/// <summary>
/// Gets the type of the view.
/// </summary>
/// <value>
/// The type of the view.
/// </value>
public override Type ViewType
{
        get
        {
                return typeof(MseMainDialogExtendedView);
        }
}

/// <summary>
/// Gets the type of the model.
/// </summary>
/// <value>
/// The type of the model.
/// </value>
public override Type ModelType
{
        get
        {
                return typeof(MseMainDialogExtendedModel);
        }
}


/// <summary>
/// Gets the typed view.
```

```
/// </summary>
private MseMainDialogExtendedView TypedView
{
        get
        {
                return this.typedView ?? (this.typedView =
                (MseMainDialogExtendedView)View);
        }
}

/// <summary>
/// Gets the specific model.
/// </summary>
private MseMainDialogExtendedModel TypedModel
{
        get
        {
                return this.typedModel ?? (this.typedModel =
                (MseMainDialogExtendedModel)Model);
        }
}

/// <summary>
/// Gets the typed data view.
/// </summary>
private MseMainDialogExtendedDataView TypedDataView
{
        get
        {
                return this.typedDataView ?? (this.typedDataView =
                (MseMainDialogExtendedDataView)TypedModel.DataView);
        }
}

#region Action Handlers

/// <summary>
/// Text action handler.
/// </summary>
/// <param name="sender">The sender.</param>
```

```
        /// <param name="args">The <see
        /// cref="com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroupEventArgs"/>
        /// instance containing the event data.</param>
        private void ExtAboutActionHandler(IActionGroup sender, IActionGroupEventArgs
        args)
        {
                var ctrl =
                MasterDialogController.CreateController(typeof(AboutViewController));
                MasterDialogController.PushController(ctrl);
        }

        #endregion Action Handlers

    }
}
```

One of the most important places in this code snippet is the following *ActionEnumType*-property:

```
/// <summary>
/// Gets the type of the action enum.
/// </summary>
/// <value>
/// The type of the action enum.
/// </value>
public override List<Type> ActionEnumType
{
    get
    {
        List<Type> aetList = new List<Type>() { typeof(ActionKeys) };
        aetList.AddRange(base.ActionEnumType);
        return aetList;
    }
}
```

The *ActionEnumType* must return a list of all *ActionKeys* in the deviation-hierarchy. This is the place where it collects all *Actionkeys* from the base class(es) and puts them together with its own *Actionkeys* into a list, which is returned to the caller.
The rest works like in the first example.

2.3.2.2 Deriving a Data View

In the data view, you select data from the database and/or get data from a global container called *VolatileDataContainer*. The *VolatileDataContainer* is used to store volatile objects.
Data objects can be added, deleted, and searched and are transported in the context of a controller from one controller to another. This gives you a simple way to bring data objects from one dialog to another so that many dialogs in a chain can add their data. In this way, the last dialog in a chain, that is, the invoice dialog gets all the data necessary to generate the invoice, write data to the database, and to commit the transaction.
In this example, you read how to select some data from the database and show it in a message box. Furthermore, you send the data to the next controller via the *VolatileDataContainer*.

2.4 Supported Extension Scenarios

Controllers are the base and are instantiated first of all (before model, view, data view and so on). You may replace controllers completely, inject them in the standard flow, or just extend them accordingly.

Recommendation

We do not recommend that you extend existing views with regards to the layout as some issues might arise with that approach. You could take this approach in some easy cases. However, if you want to modify an existing UI in depth, we recommend that you create it from scratch and use it accordingly in your extended controller.

2.4.1 Extending a Controller

If you know which controller to extend, you can hook in at the following:

```
namespace MobileSalesExtensibilityTest
{
	class ExtensibilityPlugin : IExtensibilityPlugin
	{
		/// <summary>
		/// Gets the controller extended.
		/// </summary>
		/// <param name="controllerType">Type of the controller.</param>
		/// <returns></returns>
		public Type GetControllerExtended(Type controllerType)
		{
		}
	}
}
```

> **Note**
>
> The complete Extensibility Plugin Listing can be seen in the Appendix.

If you want to extend a controller, you derive from an existing mobile sales client controller and override / change the behavior / code as required. Hence, this would look as follows:

```
namespace MobileSalesExtensibilityTest
{
	public class AboutViewExtendedController : AboutViewController
	{
		#region Overrides

		/// <summary>
		/// Gets the type of the view.
		/// </summary>
		/// <value>
		/// The type of the view.
		/// </value>
		public override Type ViewType
		{
			get
			{
				return typeof(AboutViewExtended);
			}
		}
```

```
            protected override void RegisterAction(Enum key,
com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroup ag)
            {
                  …
                  base.RegisterAction(key, ag);
            }


            protected override void DeregisterAction(Enum key,
com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroup ag)
            {
                  …
                  base.DeregisterAction(key, ag);
            }

      }
}
```

> **Note**
>
> A complete listing can also be found in the Appendix. Note the overridden *ViewType* property that shows you how to exchange the view to be displayed. In exactly the same way, you can replace a model and then a data view. In this way, you have full flexibility to replace and modify what is allowed to override.

2.4.2 Replacing a Controller

Replacing a controller works in exactly the same way as above except that you do not derive from an existing controller.

2.4.3 Injecting a Controller

Controller injection is a bit different in that you have to derive from the framework base class instead of any existing controller. Nevertheless, the basic principles remain the same. When being asked via *IExtensibilityPlugin::GetControllerExtended,* you can return your new controller, which will be used instead of the controller that should have come next.

2.4.4 Adding own Hardware Plugins

There is a folder named DeviceDrivers where any kind of hardware plugins take place. These plugins are types of sensors. We currently support implementing the following types of hardware:

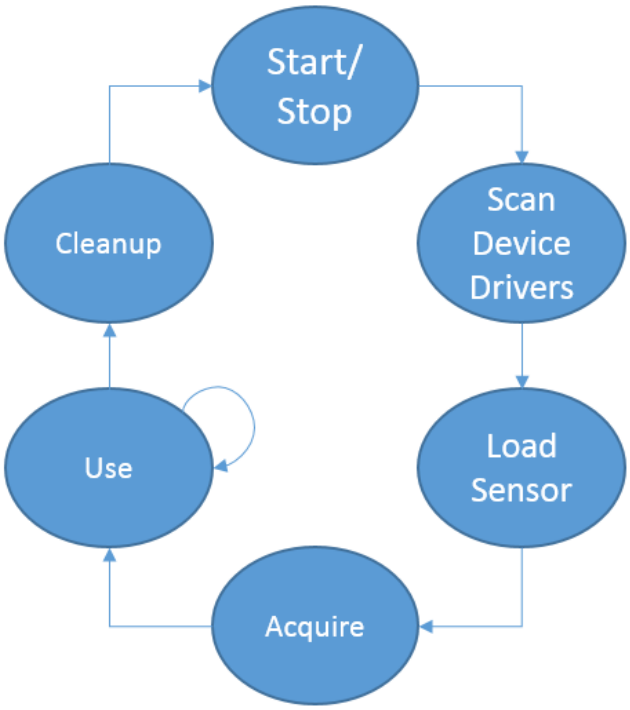
- Network
- Battery
- Scanner
- AudioInbound
- Camera
- Compass
- GPS/PositionTransducer

If you want to implement your own hardware abstraction plugin, you have to implement the interface `com.sapcd.mobileclient.hal.Interfaces.ISensorDeviceDriver`. If you need to reference any specific DLLs in order to use or access the device, you have to ensure that all these DLLs are also available in the DeviceDrivers folder.

The sensor interface comprises different aspects you have to implement and specify such as the following:

- A unique name for the hardware sensor
- An empty or prepared DeviceDriverMain() method
- A cleanup method when shutting down the sensor
- The category to which the sensor belongs (see above)
- A method TriggerSensor that handles an active sensor request
- The Enabled property that defines whether or not the sensor is in an operable state

The lifecycle of the hardware sensor during the start and shutdown of the application looks as follows:



In your code, you can acquire an instance of any sensor as follows:

1. Activate the appropriate sensor category you want to use, for example:
   ```
   DeviceDriverManager.Singleton.SetSensorDeviceCategoryEnabled(com.sapcd.
   mobileclient.hal.Interfaces.SensorDeviceHardwareCategory.Network,
   true);
   ```
2. Handle any events fired from the sensor
   ```
   DeviceDriverManager.Singleton.NetworkSensorEvent +=
   DeviceDriverManager_NetworkSensorEvent;
   ```

If you want to actively trigger the sensors for getting initial values, use the following:

```
DeviceDriverManager.Singleton.TriggerSensors();
```

You may also access the sensor by name if you want to use a specific one:

```
DeviceDriverManager.Singleton[<name>]
```

You can perform a `TriggerSensor` on it. The result will be propagated by an event like above (see 2.).

2.4.5 Application-Specific Plugins

There are the following two applications plugins, which have to be located within the applications directory (<mse install dir>\applications), that you can use and replace/extend:

- ItemProposal
- CaptureOnHand

ItemProposal interface looks as follows:

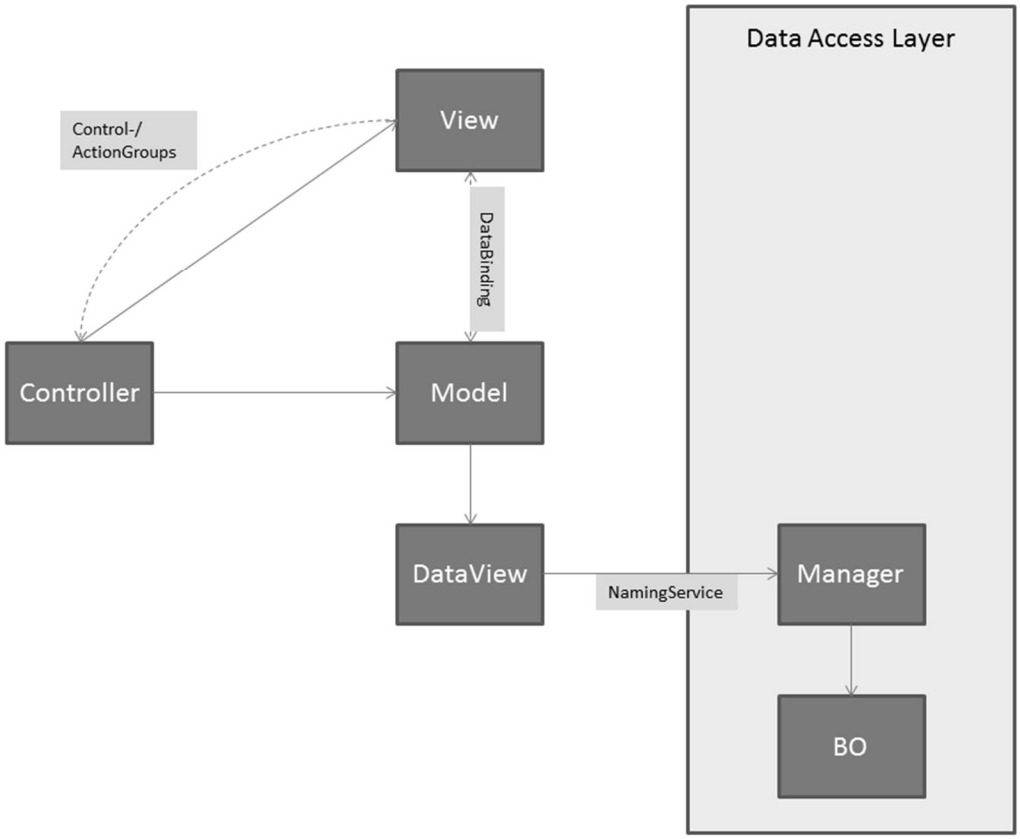
```
      /// <summary>
      /// Provides an interface to plugin item-proposal functionality.
      /// </summary>

   public interface IItemProposalPlugin : IBasePlugin
   {
            /// <summary>
            /// Calculates the proposals for a customer based on proposal
items from the backend,
            /// and the items that the customer still has in stock.
            /// </summary>
            /// <param name="visit">The visit.</param>
            /// <param name="customer">The customer.</param>
            /// <param name="proposeItems">The propose items.</param>
            /// <param name="captureOnHandItems">The capture on hand
items.</param>
            /// <returns>
            /// A list of IProposeItemBO that will be presented to the
customer as proposed delivery.
            /// </returns>
      IList<ProposeOrderItemBO> CalculateProposals(
                  VisitBO visit,
         CustomerBO customer,
         IEnumerable<ProposeOrderItemBO> proposeItems,
         IEnumerable<CaptureOnHandItemBO> captureOnHandItems);
   }
```

CaptureOnHand interface looks as follows:

```
   public interface ICaptureOnHandDAL
   {
      /// <summary>
      /// Creates the capture on hand.
      /// </summary>
      /// <param name="captureOnHand">The capture on hand.</param>
      /// <returns></returns>
      bool CreateCaptureOnHand(CaptureOnHandBO captureOnHand);

      /// <summary>
      /// Creates the capture on hand item.
      /// </summary>
```

```
        /// <param name="captureOnHandItem">The capture on hand item.</param>
        /// <returns></returns>
        bool CreateCaptureOnHandItem(CaptureOnHandItemBO captureOnHandItem);

        /// <summary>
        /// Creates the capture on hand items.
        /// </summary>
        /// <param name="captureOnHandItems">The capture on hand items.</param>
        /// <returns></returns>
        bool CreateCaptureOnHandItems(IList<CaptureOnHandItemBO>
captureOnHandItems);

        /// <summary>
        /// Checks if a capture on hand exists for the given visit.
        /// </summary>
        /// <param name="customerSalesArea">The customer sales area.</param>
        /// <returns></returns>
        bool ExistsCaptureOnHand(ICustomerSalesAreaBO customerSalesArea);

        /// <summary>
        /// Finds the capture on hand for the given visit.
        /// </summary>
        /// <param name="customerSalesArea">The customer sales area.</param>
        /// <returns></returns>
        CaptureOnHandBO FindCaptureOnHand(ICustomerSalesAreaBO
customerSalesArea);

        /// <summary>
        /// Finds the capture on hand items.
        /// </summary>
        /// <param name="customerSalesArea">The customer sales area.</param>
        /// <returns></returns>
        IList<CaptureOnHandItemBO> FindCaptureOnHandItems(ICustomerSalesAreaBO
customerSalesArea);
    }
```

2.5 Architecture



2.5.1 Controller

Namespace: `com.sapcd.mobileclient.dialog.WMForms.Controller`, `com.sapcd.mobileclient.dialog.Implementation.Controller`

Every controller representing a new screen has to derive from *MobileDialogController*, which is the base class for developing a new controller. In that controller, you are able to override several methods and properties.

The sequence of the methods called during instantiation is as follows:

1. ctor
2. CreateModel
3. <model>.CreateDataView
4. CreateView
5. Initialize
6. RegisterControlGroups
7. RegisterActionGroups
8. <view>.RegisterDataView
9. InitializationFinished

You may hook-in at every point in this sequence and perform actions or execute program code that suits your current requirements.

You are able to define your model and view by overriding the following properties:

- ModelType
- ViewType

The data view to be created can be defined by overriding the following model class property:

- DataViewType

Following this, you can override all components and replace them with your own classes as required.

> **Note**
>
> The main entry point is always the controller.

2.5.2 Model

Namespace: `com.sapcd.mobileclient.dialog.Implementation.Model`

The model is intended to store data that is relevant just for the lifetime of the screen. It should be destroyed when the screen is left.

2.5.3 Data View

Namespace: `com.sapcd.mobileclient.dialog.Implementation.Model`

The data view is used for retrieving business related data from the database using data access layer manager classes.

> **Note**
>
> Refer to the public API of the data view in order to see what can be overridden. In 90% of all cases you want to override the *SelectData* method where your data selection starts.

2.5.4 Action Groups

The concept of these groups is to decouple UI controls from actions, which they trigger in the controller. You just register one or multiple controls and can combine them to trigger the same action. If you just want to control some UI controls, then a *ControlGroup* is the class of choice since you may then enable or disable the controls accordingly.

The registering of action and control groups is done transparently. Make sure you consider the following snippets and overrides in your controller:

- ActionEnumType

  for example, (notice: considers base actions, too):

```
/// <summary>
/// Gets the type of the action enum.
/// </summary>
/// <value>
/// The type of the action enum.
/// </value>
public override List<Type> ActionEnumType
{
	get
	{
		List<Type> aetList = new List<Type>() { typeof(ActionKeys)
};
		aetList.AddRange(base.ActionEnumType);
		return AELIst;
	}
}
```

- RegisterAction

```
/// <summary>
/// Registers the action.
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
protected override void RegisterAction(Enum key, IActionGroup ag)
{
	ActionKeys tmpKey = (ActionKeys)key;
	switch (tmpKey)
	{
		case ActionKeys.AboutExt:
		{
			ag.Action += ExtAboutActionHandler;
			break;
		}
	}
	base.RegisterAction(key, ag);
}
```

- DeregisterAction

```
/// <summary>
/// De-Registers the action.
/// </summary>
/// <param name="key">The key.</param>
/// <param name="ag">The ag.</param>
protected override void DeregisterAction(Enum key, IActionGroup ag)
{
	ActionKeys tmpKey = (ActionKeys)key;
	switch (tmpKey)
	{
		case ActionKeys.AboutExt:
		{
			ag.Action -= ExtAboutActionHandler;
			break;
		}
	}
	base.DeregisterAction(key, ag);
}
```

Using *ControlGroups* works according to the following same principles.

- Define ControlGroup enum – **do not mix** with action enum

- `protected override void RegisterControlGroup(Enum key, IControlGroup ag)`
- `protected override void DeregisterControlGroup(Enum key, IControlGroup ag)`

2.5.5 Views

Recommendation

We do not recommend that you derive from a view. If you want to heavily modify the layout of a view, we recommend that you build it from scratch.

When to derive:

- Inject a menu item in dialog menu
- Change text or color
- Inject a control without modifying the rest of the UI

When not to derive:

- Heavy layout changes / manipulations
- Re-arrangement of controls

2.5.6 Managers and Data Access Layer

2.5.6.1 General

The access to business data is done using a manager class, which has to implement either a standard interface or a completely new manager for data access.

Note

See the Appendix for existing standard interfaces for the data access layers (DALs) and managers.

2.5.6.2 DAL Component

If you want to modify extend or replace a manager or DAL class, you have to hook-in at data view level.

2.5.6.2.1 Extend or Modify a DAL

You can extend a DAL by deriving from it and overriding intended methods. You may call the base method if required or replace the functionality completely.

2.5.6.2.2 Replace a DAL

You can replace a DAL by deriving from intended framework classes and by modifying the corresponding data view as well where the DAL is created. Instead of using the standard DAL, you can instantiate your own DAL.

> **Note**
>
> You have to register your DAL at *MseNamingService*. Therefore, you have to call a corresponding register method as follows in there: `this.Register<IOwnDAL>(typeof(OwnDAL));`

Afterwards, you can instantiate your DAL in the following way (standard example DAL):

```
using (MseNamingContext context = new MseNamingContext())
{
	OwnDalObj = context.GetInstance<IOwnDAL>();
}
```

Your DAL must be derived as follows:

```
/// <summary>
/// The DAL implementation for all customer related objects.
/// </summary>
public class OwnDAL : DataAccessBase, IOwnDAL
{

}
```

2.5.6.3 Manager Component

The manager component is created, registered and extended, modified, or replaced like a DAL component.

```
this.Register<IOwnManager>(typeof(OwnManager));

using (MseNamingContext context = new MseNamingContext())
{
	OwnMgrObj = context.GetInstance<IOwnManager>();
}
```

> **Note**
>
> The main difference between a DAL and a manager is that a manager is used for high-level logical data retrieval whereas a DAL is mainly used for direct database access.

3 Pricing Requirements and Formula Routines

Requirements and formulas are routines provided to customize the determination of condition records and the evaluation of conditions in pricing. These routines can be maintained in SAP ERP in *Maintain Requirements and Formulas* (transaction **VOFM**). The routines in the back end of SAP ERP are ABAP code written in forms.

SAP Direct Store Delivery supports the usage of requirement and formula routines in offline pricing performed on mobile devices. In order to facilitate the usage of these routines, the offline pricing engine (the component of *SAP Direct Store Delivery* responsible for pricing) contains implementation for some widely used standard SAP ERP routines delivered by SAP and offers provision for customers to hook the implementation of own routines in pricing calculation logic.

> **Note**
>
> The SAP namespace is 1-599 and the customer namespace is 600-999 for requirement, condition value formula, and condition base value formula and scale base formula routines. For group key formula routines, the SAP namespace is 1-49 and the customer namespace is 50-99.

3.1 Standard Routines

3.1.1 Requirements

Requirements are rules that determine when a particular action should take place and that aid system performance by defining rules that eliminate unnecessary accesses to the database. For example, the user may have listed condition type **PR00** first to determine the price. If a requirement has been assigned to **PR00** in the pricing procedure, the system checks whether or not it is met. If it is met, the system continues to look for prices using the assigned access sequence. If it is not met, the system skips that condition type and goes to the next level of the pricing procedure. The same holds true for the individual levels of the access sequence. If a requirement is assigned, the access is only made if the requirement is met.

In SAP ERP, you can maintain requirements on pricing procedure step/counter level and on condition access step level. You do this in Customizing (transaction **SPRO**) for *Sales and Distribution* under *Basic Functions* è *Pricing* è *Pricing Control:*

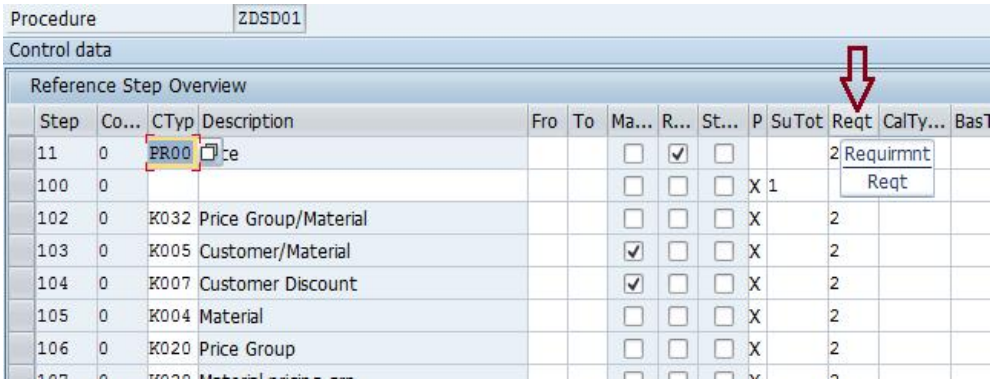


Figure 1: Requirement Assigned at Pricing Procedure Level

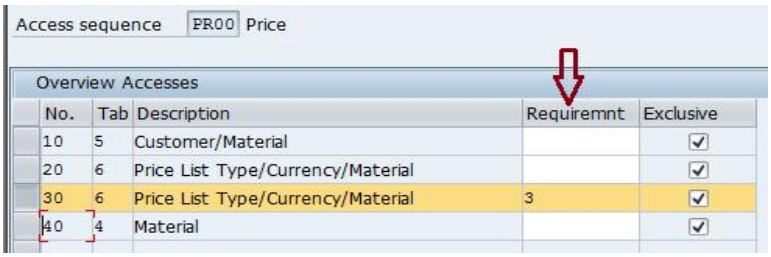


Figure 2: Requirement Assigned at Access Sequence Level

3.1.2 Condition Value Formulas

Condition value formulas are available to influence the condition value that is displayed for a particular condition type or value line in the pricing procedure. Condition value formulas can also be used to compute values that should appear as value lines in the pricing procedure. A condition value formula is assigned to a condition type in the pricing procedure.

If group condition processing for that condition type is enabled, then the calculation is done at pricing document/ header level as well as on item level.

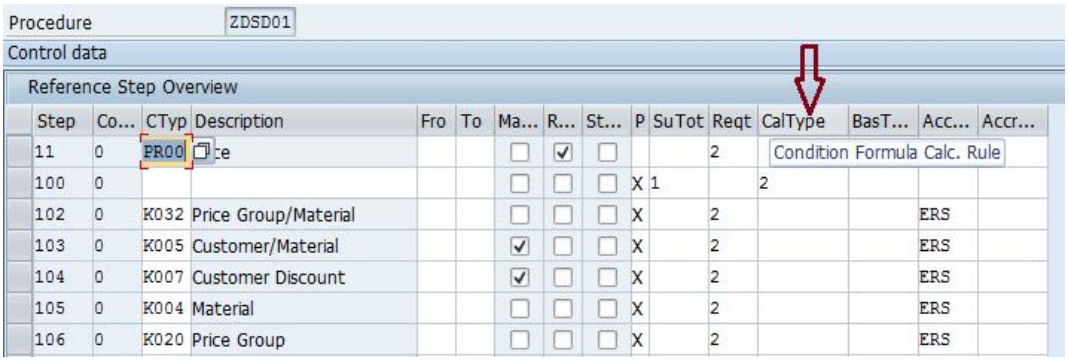


Figure 3: Condition Value Formula Assigned at Pricing Procedure Level

3.1.3 Condition Base Value Formulas

Condition base value formulas can be used to change the automatically calculated base value of a condition and to influence the condition basis to which the pricing condition rate will be applied. In standard pricing, the system applies the condition rate to the quantity in the sales document.

A condition base value formula is commonly used to determine the base for distributing header discounts/surcharges to the sales document line items. A condition base value formula is assigned to a condition type in the pricing procedure.

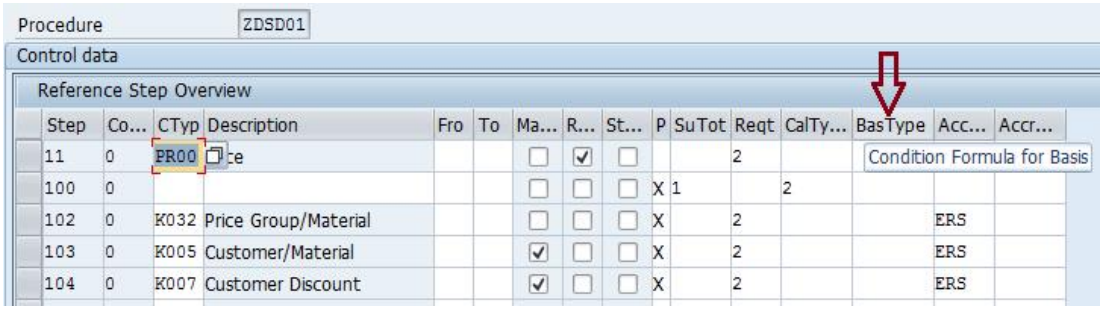


Figure 4: Condition Base Value Formula Assigned at Pricing Procedure Level

3.1.4 Scale Base Formulas

A scale base formula can be used to alter the value that the system uses to read the scales in a condition record during pricing. This user exit is called after the calculation of the condition scale base value for a pricing condition. If group condition processing for that condition type is enabled, then the calculation is also done at pricing document/header level as well as on item level.
In standard pricing, the system reads the scale in a pricing condition record using, for example, the quantity, value, or weight, of the document depending on the type of scale that has been defined.

Example

The price for product A is as follows:

- From 0 cases - $50 per case
- From 100 cases - $45 per case
- From 500 cases - $40 per case

If the customer orders 100 cases of product A, the system reads the pricing scale using 100 cases and determines the appropriate price, $45 per case. Using a scale base formula, it is possible to alter the value of 100 prior to the scale being read.

Scale base formulas are assigned to pricing condition types in the SAP ERP configuration.

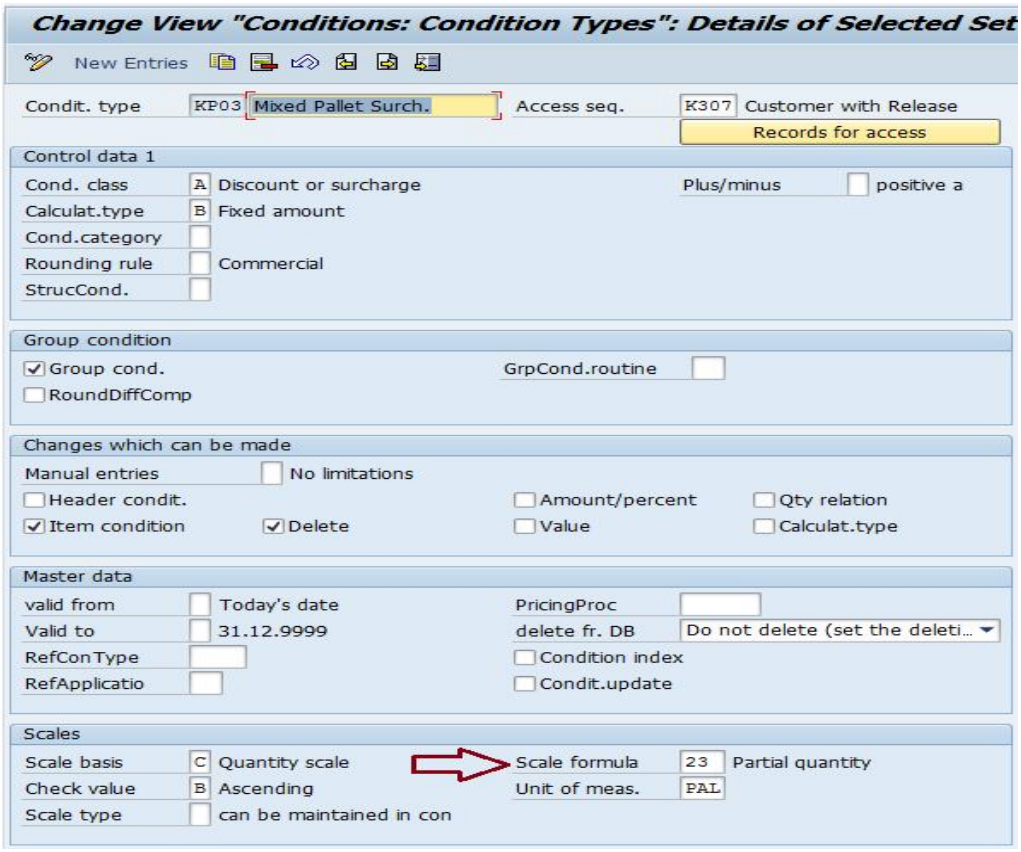


Figure 5: Scale Base Formula Assigned at Condition Type Level

3.1.5 Group Key Structure Formulas

This seldom used exit influences the grouping of group conditions (conditions that are processed together over more than one item). A structure of group key formula can be used to influence the basis the system uses when reading the scale of a group condition. Group conditions are used to cumulate quantities from more than one sales document line item to read pricing scales.

Example

When pricing a particular sales document line item, you want the system to not just consider the quantity of the current line item but the sum of the quantities of all line items that share the same material pricing group as the current line item.

The formula is assigned to a group condition type in Customizing.

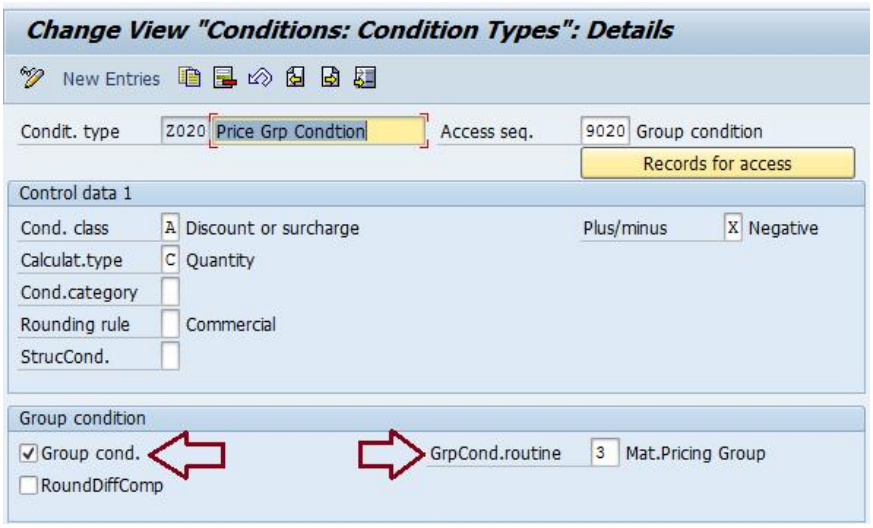


Figure 6: Group Key Structure Formula Assigned at Condition Type Level

3.2 Customer Routines

To implement own routines, you must create a project of type class library with the following technical characteristics;

- Project language C#
- Runtime environment .NET framework 3.5
- Project type Class Library
- Include the following DLLs as project dependency:
  - SAPCD.MobilePricing.Interfaces
  - SAPCD.MobilePricing.Common

> **Note**
>
> The above listed DLLs can be obtained after installing the CAB file containing the *SAP Direct Store Delivery* front end application. The path to these DLLs is {**`Storage Disk for handheld device}\\Program Files\mse\applications`**

The following prerequisites must be observed for all types of customer routine implementations:

1. Each routine implementation must be done in an own class within the project created with above technical characteristics.
2. The class containing the routine logic must implement an SAP provided interface specific for the routine type (see the subsequent sections for the name of the interface to be used for each routine type).
3. The customer-specific routine should also be implemented in the back end.
4. Routine numbers should fall within the allowed range for the customer namespace.

The DLL containing customer routines must be placed in the same directory (that is, {**`Storage Disk for handheld device}\\Program Files\mse\applications`**) along with all pricing related standard DLLs.

In addition to implementing the routines and placing the DLL in the requisite location, the existence of customer routine implementation needs to be indicated to the pricing engine. This is achieved by maintaining the corresponding configuration property in the PricingEngine.config file. This file is located along with all other configuration files in the following directory:

{**`Storage Disk for handheld device}\\Program Files\mse\config`**

The pricing engine checks the PricingEngine.config file during initialization. If there is a valid value maintained for the UserExits.UserExitAssembly' property, it loads this assembly dynamically using the assembly name provided as value.

> **Example**
>
> UserExits.UserExitAssembly=ZZ.MobilePricing.UserExits.dll

3.2.1 Implementing Requirement Routine

In order to use a customer defined requirement routine during price calculations, the IRequirement interface contained in the SAPCD.MobilePricing.Common DLL must be implemented. This interface contains a single method definition namely ‘CheckRequirement’. The logic to check whether a pricing line item satisfies the requirement must be provided in this method.

When the pricing engine encounters a requirement number within the customer namespace specified on pricing procedure step / counter level or on condition access step level, it loads the customer’s implementation. In case of missing or incorrect configuration of customer routines, the pricing engine logs an error.

Each requirement routine implementation along with the corresponding routine number must be listed in the PricingEngine.config file in the following format:

Example

```
UserExits.Requirement=3
UserExits.Requirement.0=601,ZZ.MobilePricing.UserExits.Requirement601
UserExits.Requirement.1=602,ZZ.MobilePricing.UserExits.Requirement602
UserExits.Requirement.2=902,ZZ.MobilePricing.UserExits.Requirement902
```

The configuration property ‘`UserExits.Requirement`’ specifies the total number of customer implemented requirement routines to be loaded by the pricing engine. Based on this number, the pricing engine looks for configuration keys containing the routine number and the fully qualified class name containing the routine implementation. For instance, in the above example, since ‘3’ has been specified, the pricing engine looks for exactly three routines with the following keys `UserExits.Requirement.0, UserExits.Requirement.1,` and `UserExits.Requirement.2.`

In case of missing or incorrect configuration, the loading of the routine in question is skipped and the pricing engine logs the error and proceeds with further initialization steps.

3.2.2 Implementing Condition Value Formula

In order to use a customer defined condition value formula, the IConditionValueFormula interface contained in the SAPCD.MobilePricing.Common DLL must be implemented. This interface contains a single method definition namely ‘OverwriteConditionValue’. The logic to influence the condition value during price calculation must be provided in this method.

This method is called after the calculation of the condition value for the pricing condition. The pricing engine looks for the customer implementation when a formula number in the customer namespace is defined for a condition type. In case of missing or incorrect configuration of customer formulas, the pricing engine logs an error.

Each condition value formula implementation along with the corresponding formula number must be listed in the PricingEngine.config file in the following format:

Example

```
UserExits.ConditionValueFormula=3
UserExits.ConditionValueFormula.0=601,ZZ.MobilePricing.UserExits.ConditionValueFormula601
UserExits.ConditionValueFormula.1=602,ZZ.MobilePricing.UserExits.ConditionValueFormula602
UserExits.ConditionValueFormula.2=919,ZZ.MobilePricing.UserExits.ConditionValueFormula919
```

The configuration property ‘`UserExits.ConditionValueFormula`’ specifies the total number of customer implemented condition value formulas to be loaded by the pricing engine. Based on this number, the pricing engine looks for configuration keys containing the formula number and the fully qualified class name containing the formula implementation. The logic for loading formula implementations based on the number specified in the ‘`UserExits.ConditionValueFormula`’ property is exactly the same as described in section 3.2.1 for loading requirement routines.

3.2.3 Implementing Condition Base Value Formula

In order to use a customer defined condition base value formula, the IConditionBaseFormula interface contained in the SAPCD.MobilePricing.Common DLL must be implemented.

This interface contains two method definitions namely ‘GetFormulaType’ and ‘OverwriteConditionBase’. The logic to influence the condition base value during price calculation must be provided in the ‘OverwriteConditionBase’ method.

The ‘GetFormulaType’ method provides an indicator to pricing logic when exactly the formula should be called. Some Condition Base formulas overwrite the condition base value to be used as basis for further processing, for example, unit of measure conversion, and some formulas operate on finally determined base value to round it off.

The formula type indicator is checked during condition base value determination and the formula implementation is called accordingly. The possible values are defined in an enumeration called ‘FormulaType’ (included in the SAPCD.MobilePricing.Interfaces DLL) and are described below:

```
/// <summary>
/// Default Category of Formula Implementation. This type of formulas
does not impact the condition base value directly
/// but rather modify other parameters e.g. condition exclusion
indicator etc.
/// </summary>
None,

/// <summary>
///  This formula should be called after the UoM conversion and other
determination logic has been executed.
 ///  Example: Condition Base Formula 22, which rounds off the final
calculated value.
  /// </summary>
ModifiesValue,

/// <summary>
///  This formula should be called before the UoM conversion and other
value determination logic is executed.
///  Example: Condtion Base Formula 13, which provides Net Weight as
basis for further calculation.
/// </summary>
SubstitutesValue
```

Each condition base value formula implementation along with the corresponding formula number must be listed in the PricingEngine.config file in the following format:

Example

UserExits.ConditionBaseFormula=2

UserExits.ConditionBaseFormula.0=601,ZZ.MobilePricing.UserExits.ConditionBaseFormula601

UserExits.ConditionBaseFormula.1=602,ZZ.MobilePricing.UserExits.ConditionBaseFormula602

The logic for loading formula implementations based on the number specified in the ‘UserExits. ConditionBaseFormula’ property is exactly the same as described in section 3.2.1 for loading requirement routines.

3.2.4 Implementing Scale Base Value Formula

In order to use a customer defined condition value formula, the IConditionScaleBaseFormula interface contained in the SAPCD.MobilePricing.Common DLL must be implemented. This interface contains a single method definition namely ‘OverwriteScaleBase’. The logic to alter the value that the system uses to read the scales in a condition record during pricing must be provided in this method.

Each scale base formula implementation along with the corresponding formula number must be listed in the PricingEngine.config file in the following format:

Example

UserExits.ScaleBaseFormula=2

UserExits.ScaleBaseFormula.0=601,ZZ.MobilePricing.UserExits.ScaleBaseFormula601

UserExits.ScaleBaseFormula.1=602,ZZ.MobilePricing.UserExits.ScaleBaseFormula602

The logic for loading formula implementations based on the number specified in property ‘UserExits. ScaleBaseFormula’ is exactly the same as described in section 3.2.1 for loading requirement routines.

3.2.5 Implementing Group key Formula

In order to use a customer defined group key structure formula, the IGroupConditionKeyFormula interface contained in the SAPCD.MobilePricing.Common DLL must be implemented. This interface contains a single method definition namely ‘SetGroupConditionKey’. This method is called after the key of a group condition is determined on the pricing document level.

Each group key formula implementation along with the corresponding formula number must be listed in the PricingEngine.config file in the following format:

Example

UserExits.GroupKeyFormula=2

UserExits.GroupKeyFormula.0=51,ZZ.MobilePricing.UserExits.GroupKeyFormula51

UserExits.GroupKeyFormula.1=52,ZZ.MobilePricing.UserExits.GroupKeyFormula52

The logic for loading formula implementations based on the number specified in the ‘UserExits.GroupkeyFormula’ property is exactly the same as described in section 3.2.1 for loading requirement routines.

4 Mobile Reports: Templates, Keys, and Tables

The reports in the SAP Direct Store Delivery front end component are based on templates. In order to meet your specific needs, you can edit the templates to some extent. The files are located at <InstallDir>\applications\printing\templates. <InstallDir> is the installation directory of the mobile application. If you want to use images in the reports, save them under <InstallDir>\applications\printing\resources.

The following report templates are part of the solution:

Report Name/Transition	Template Name
After check-out/check-in	COCIDifferences_Report
Total inventory differences	COCITotalDifferences_Report
After payment confirmation	Collection_Receipt
After confirmation of return delivery	CreditNote_Report
After confirmation of delivery w/o invoice	DeliveryNote_Report
After confirmation of delivery with empties w/o invoice	DeliveryNote_Report_With_Empties_Section
From visit screen	Empties_Balance_Report
Expenses	Expense_Summary
Inventory	Inventory_Report
After confirmation of delivery	Invoice_Report
After confirmation of delivery with empties	Invoice_Report_With_Empties_Section

Report Name/Transition	Template Name
After confirmation of order	Order_Report
After confirmation of order with empties	Order_Report_With_Empties_Section
Presettlement	Presettlement_Report
Route performance	Route_Performance
After confirmation of delivery with empties	Valuated_Delivery_Note_Report_With_Empties_Section
Visit details	Visit_List_Details_Report
Visit discrepancies	Visit_List_Discrepancies_Report
Visit overview	Visit_List_Overview_Report
From visit screen	VisitPickList_Report
Pending collections Is only supported as of SP01.	PendingCollection_Report

4.1 Adjusting Reports

You can edit the report template files using a standard text editor. If a new template file is created for a particular report, its name must be maintained in SAP ERP in Customizing for the mobile reports. More information about Customizing for reports can be found in the Configuration Guide, which is located on the SAP Service Marketplace at http://service.sap.com/instguides.

4.2 Report Syntax

Each report file supports the following main syntax elements:

- Plain text – Hello this is plain text!
- Keys to be replaced by data – {CompanyName}
- Keys to be replaced by content - images – {IMG}myImage.bmp{!IMG}
- Keys to be replaced by content - lists – {LIST}myTable,COLUMNS (1|2|3), ALIGN(L|L|R){!LIST}
- Keys to be replaced by content - signatures – {SIG}mySignature{!SIG}
- Layout – empty lines, plain-text / value spacing
- Format lines to specify spacing for each content line – | | |

The keys that are entered into a report file are replaced with the corresponding content by the application. The data coming from the application is treated like a key-value pair, where the variables name is the key and the content is the value, that is, a text, an amount of money, a number, and so on.

Example

companyName - SAP SE (companyName = key, SAP SE = value)

The list-syntax ({LIST} supports different attributes, which are separated by commas:

- Name – Mandatory, for example, myTable
- HDR – Optional, indicating that the column headers should be printed
- COLUMNS – Optional, indicating which columns of the table should be printed
- ALIGN – Optional, alignment of the column values (R-right, L-left, C-Center)

When printing images using the image-syntax ({IMG}), the following requirements need to be taken into account:

- Black / white – 1 bit colordepth
- Windows Bitmap (.bmp) format
- Placed on file system in defined folder (\applications\printing\resources)

When editing a file, the following rules must be observed:

- All content lines in a template file need to have own format line below
- All format lines (also including those for lists) need to begin and end with “|”
- Format lines do not appear on the printout
- To print an empty line, just put an empty line in the template file
- All content of a template file is formatted using the corresponding format line
- No alignment of content except left-alignment
- In a format line tab, tabs and / or spaces can be used to define the spacing
- Plain text + key / value is interpreted as one cell / column

Example

Template file

```
{CompanyName}
|      |
{CompanyCity}
|             |

Date: {Date}
|               |
Driver name: {DriverName}
|                    |
{LIST}myTable,HDR,COLUMNS(0|1|2){!LIST}
|             |             |             |
{IMG}sap.bmp{!IMG}

{SIG}mySignature{!SIG}
```

Corresponding output

```
SAP
Walldorf

Date: 24.12.2013
Driver name: Test Driver
Item          Item number         Quantity
1             1180                3
2             1191                5
3             2345                1
```

4.3 Report Keys, Tables, and Blocks

Besides static texts, the reports consist mainly of keys fields, tables, and blocks.

Key Fields

A “key” field is a simple placeholder in a template that gets substituted with the actual content of that field. The default separator pair for these key fields are the “{“ and “}” characters. This separator pair can be configured in the Printer.config. A key field can be used in a report template (for example, “{TourId}”), which then gets replaced by the actual value (for example, “S0010000123”).

Multiple report templates make use of data that is maintained in the application Customizing. These fields are documented in the following format: MSE_CUSTOMIZING “<KEY>” - <KEYVALUE>. For example, the driver’s company is documented with: MSE_CUSTOMIZING “COMPANY_NAME” – Sales Organization, whereas the field Sales Organization is dependent on the current tour data.

Table Fields

Table fields begin with the separator “{LIST}”, followed by the table identifier, and end with “{!LIST}”. Additionally, tables can enable a header for each column with the key word “HDR”. Those columns can be included individually by using the “COLUMNS()” keyword. Also, it is possible to align these columns by using the “ALIGN()” keyword. For example, a full table definition in a report would look as follows:
{LIST}<identifier>,HDR,COLUMNS(0|1|2|4),ALIGN(L|L|L|R){!LIST}

In this example, the table has an enabled header and four columns with alignment. However, the fourth column (ID 3) is not included in the table.

Block Fields

Some reports make use of the “block” feature of the reporting engine. Basically, this feature provides a report within another report. Hence, reports can be built more dynamically by incorporating sub-reports. Block reports keys are used like tables and begin with the separator “{BLOCK}”, followed by the block’s key identifier, and end with “{!BLOCK}”.

These block reports can be used like normal reports and make use of normal key fields and table fields.

4.4 Available Fields per Report

4.4.1 General Report Data

The values listed here can be used in every report across the whole application.

4.4.1.1 Key Fields

Key	MBO	Description
TourId	MSE_TOUR_HD.TOUR_ID	The tour ID of the current tour
Driver	MSE_TOUR_HD.DRIVER_NAME1	The current driver
Route	MSE_TOUR_HD.ROUTE	The current route
Vehicle	MSE_TOUR_HD.VEHICLE_ID	The current vehicle
Date	-	The current date, for example, 05/30/13
DateTime	-	The current time, for example, 16:32
CompanyName	MSE_CUSTOMIZING "COMPANY_NAME" – *Sales Organization*	The name of the driver's company
CompanyAddress	MSE_CUSTOMIZING "COMPANY_ADDRESS" – *Sales Organization*	The address of the driver's company
CompanyVAT	MSE_CUSTOMIZING "COMPANY_VAT" – *Sales Organization*	VAT of the driver's company
CustomerSignature		
DriverSignature		
CheckerSignature		

4.4.1.2 Table Fields

None

4.4.1.3 Block Fields

None

4.4.2 COCI Differences Report Data

The template file is “COCIDifferences_Report.txt”. This report supports signatures for the “Checker” and “Driver” roles.

4.4.2.1 Key Fields

Key	MBO	Description
ReportTitle	-	Localized header dependent on whether this report is check out or check in.
cociItemsByInvType	-	Block key, see 4.4.2.3 Block Fields

4.4.2.2 Table Fields

None

4.4.2.3 Block Fields

The template for this block report is “COCIDifferences_Report_COCIItemsByInvType_Block.txt”. This block is created for each check-out / check-in item grouped by material number and UOM.

4.4.2.3.1 Key Fields

Key	MBO	Description
itemGroupTitle	-	Localized header dependent on the counting group (Empties, Free, Reserved, or Blocked).

4.4.2.3.2 Table Fields

The table key is “cociTable”.

Header	MBO	Description	Table ID
MatNo	MSE_MATERIAL_HD.MAT_NO	Number of the material	0
MatDesc	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Description of the material	1
MatUom	MSE_COCI_ITEM.UOM	UOM of the material/COCI item	2
Planned Qty	MSE_COCI_ITEM.PLAN_QTY	Planned quantity: sum of all items	3
Actual Qty	MSE_COCI_ITEM.ACT_QTY	Actual quantity: sum of all items	4
Difference Qty	-	Actual – Planned quantity (of fields above)	5
Reason Code			6
Inv Type	MSE_COCI_ITEM.COCI_REASON	Reason code of the COCI item	7

4.4.3 COCI Total Differences Report Data

The template file is “COCITotalDifferences_Report.txt”. This report supports signatures for the “Checker” and “Driver” roles.

4.4.3.1 Key Fields

Key	MBO	Description
InventoryItemsByInvType	-	Block key, see 4.4.3.3 Block Fields.

4.4.3.2 Table Fields

None

4.4.3.3 Block Fields

The template for this block report is “COCITotalDifferences_Report_COCIItemsByInvType_Block.txt”. This block is created for each inventory item grouped by material number.

4.4.3.3.1 Key Fields

Key	MBO	Description
itemGroupTitle	-	Localized header dependent on the counting group (Empties, Free, Reserved, or Blocked).

4.4.3.3.2 Table Fields

The table key is “inventoryItemTable”.

Header	MBO	Description	Table ID
MatNo	MSE_MATERIAL_HD.MAT_NO	Number of the material	0
MatDesc	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Description of the material	1
MatUom	MSE_MATERIAL_SALESORG.SALES_UOM / MSE_MATERIAL_HD.BASE_UOM	UOM of the material in sales UOM or base UOM	2
Check-Out Qty	MSE_COCI_ITEM.ACT_QTY	Sum of all quantities for a material during check-out	3
Reload Qty	MSE_COCI_ITEM.ACT_QTY	Sum of all reloaded quantities for a material (reload check-in – reload check-out)	4
Adjustment Qty	MSE_COCI_ITEM.ACT_QTY	Sum of all adjusted materials in inventory	5
Delivered Qty	MSE_DELIVERY_ITEM.DACT_QTY / FREE_QTY / RESERVE_QTY	Sum of the delivery items	6

Header	MBO	Description	Table ID
Returned Qty	MSE_DELIVERY_ITEM.DACT_QTY / FREE_QTY / RESERVE_QTY	Sum of the returned items	7
Check-In Qty	MSE_COCI_ITEM.ACT_QTY	Sum of the checked-in items	8
Difference Qty	-	Sum of the above quantities: difference = check-out + reload - adjustment - delivered + returned - check-in	9

4.4.4 Inventory Report Data

The template file is “Inventory_Report.txt”.

4.4.5 Key Fields

Key	MBO	Description
InventoryItemsByInvType	-	Block key, see 4.4.7 Block Fields.

4.4.6 Table Fields

None

4.4.7 Block Fields

The template for this block report is “Inventory_Report_InventoryItemsByInvType_Block.txt”. This block is created for each inventory item grouped by material number.

4.4.7.1 Key Fields

Key	MBO	Description
itemGroupTitle	-	Localized header dependent on the counting group (Empties, Free, Reserved, or Blocked).
uom	-	Base UOM
qtySum	-	Actual quantity of item

4.4.7.2 Table Fields

The table key is “inventoryItemTable”.

Header	MBO	Description	Table ID
MatNo	MSE_MATERIAL_HD.MAT_NO	Material number	0
MatDesc	MSE_MATERIAL_HD.MAT_DESC1	Material description	1
MatUom	MSE_STOCK.UOM_CURRENCY	Material UOM	2
Qty	MSE_STOCK.QTY	Sum of materials for this UOM	3

4.4.8 Partner Report Data

All keys and tables shown in this section are available for all partner function related reports.

4.4.8.1 Key Fields

Key	MBO	Description
billToName	MSE_CUSTOMER_HD.NAME	Customer name
billToStreet	MSE_CUSTOMER_HD.STREET	Customer street
billToCity	MSE_CUSTOMER_HD.CITY	Customer city
billToPCode	MSE_CUSTOMER_HD.PCODE	Customer postal code
billToCityAndPCode	MSE_CUSTOMER_HD.CITY / PCODE	Customer city and postal code

Key	MBO	Description
billToCountry	MSE_CUSTOMER_HD.COUNTRY	Customer country
billToCountryCode	MSE_CUSTOMER_HD.COUNTRY_CODE	Customer country code
billToNumber	MSE_CUSTOMER_HD.CUST_NO	Customer number
billToVAT	MSE_CUSTOMER_HD.VATREG_NO	Customer VAT
shipToName	MSE_CUSTOMER_HD.NAME	Customer name
shipToStreet	MSE_CUSTOMER_HD.STREET	Customer street
shipToCity	MSE_CUSTOMER_HD.CITY	Customer city
shipToPCode	MSE_CUSTOMER_HD.PCODE	Customer postal code
shipToCityAndPCode	MSE_CUSTOMER_HD.CITY / PCODE	Customer city and postal code
shipToCountry	MSE_CUSTOMER_HD.COUNTRY	Customer country
shipToCountryCode	MSE_CUSTOMER_HD.COUNTRY_CODE	Customer country code
shipToNumber	MSE_CUSTOMER_HD.CUST_NO	Customer number
shipToVAT	MSE_CUSTOMER_HD.VATREG_NO	Customer VAT
soldToName	MSE_CUSTOMER_HD.NAME	Customer name
soldToStreet	MSE_CUSTOMER_HD.STREET	Customer street
soldToCity	MSE_CUSTOMER_HD.CITY	Customer city
soldToPCode	MSE_CUSTOMER_HD.PCODE	Customer postal code
soldToCityAndPCode	MSE_CUSTOMER_HD.CITY / PCODE	Customer city and postal code
soldToCountry	MSE_CUSTOMER_HD.COUNTRY	Customer country
soldToCountryCode	MSE_CUSTOMER_HD.COUNTRY_CODE	Customer country code
soldToNumber	MSE_CUSTOMER_HD.CUST_NO	Customer number
soldToVAT	MSE_CUSTOMER_HD.VATREG_NO	Customer VAT
payerName	MSE_CUSTOMER_HD.NAME	Customer name
payerStreet	MSE_CUSTOMER_HD.STREET	Customer street
payerCity	MSE_CUSTOMER_HD.CITY	Customer city

Key	MBO	Description
payerPCode	MSE_CUSTOMER_HD.PCODE	Customer postal code
payerCityAndPCode	MSE_CUSTOMER_HD.CITY / PCODE	Customer city and postal code
payerCountry	MSE_CUSTOMER_HD.COUNTRY	Customer country
payerCountryCode	MSE_CUSTOMER_HD.COUNTRY_CODE	Customer country code
payerNumber	MSE_CUSTOMER_HD.CUST_NO	Customer number
payerVAT	MSE_CUSTOMER_HD.VATREG_NO	Customer VAT

4.4.8.2 Table Fields

The table key is “soldToShipToTable”.

Header	MBO	Description	Table ID
ColSoldTo	· No MBO · MSE_CUSTOMER_HD.CUST_NO · MSE_CUSTOMER_HD.CUST_NAME · MSE_CUSTOMER_HD.STREET · MSE_CUSTOMER_HD.CITY & MSE_CUSTOMER_HD.PCODE · No MBO · MSE_CUSTOMER_HD.VATREG_NO	Fixed table rows displaying the Sold-To information of a customer in the following order: · Column label (localized) · Customer number · Customer name · Street · City with postal code · VAT label (localized) · VAT	0
ColShipTo	See above	Same information as above but for Ship-To customer	1

4.4.9 Collection Report Data

The template file is “Collection_Receipt.txt”. This report supports signatures for the “Customer” and “Driver” roles. Also, there are other key fields and tables available for this report (see 4.4.8 Partner Report Data)..

4.4.9.1 Key Fields

Key	MBO	Description
CollectionDate	-	Date of the collection
NumberOfPayments	-	The number of all used payments in this collection
PaymentsTotal	MSE_PAYMENT.AMOUNT	The sum of all payments in this collection
CollectionNumber	MSE_COLLECTION.COLLECT_ID	The official document number
currency	MSE_CUSTOMIZING “CURRENCY” – “EUR”	The customized currency for this shipment

4.4.9.2 Table Fields

The table key is “payments”.

Header	MBO	Description	Table ID
Payment Type	MSE_CUSTOMIZING “PY_C_TYPE” *Payment Type*	The payment type based on the payment code	0
Bank/Card Type	MSE_PAYMENT.BK_KEY / PY_C_TYPE / EMSE_CUSTOM1	Either the bank name or the card type based on the current payment or the first field of a custom payment	1
Check/Card No.	MSE_PAYMENT.CHECK_NO / ACC_KEY / PY_C_NUM / EMSE_CUSTOM2	Check number, account number, or credit card number based on the current payment or the second field of a custom payment	2
Additional	MSE_PAYMENT.PY_C_EXP / EMSE_CUSTOM3	The credit card’s expiration date or the third field of a custom payment	3
Amount	MSE_PAYMENT.AMOUNT	The amount of the payment	4
Cancel	MSE_PAYMENT.CANCEL_FLAG	Whether the payment is canceled or not	5
Reason	MSE_PAYMENT.CAN_REV_REAS	Cancel reason	6

Header	MBO	Description	Table ID
Col.ID	MSE_PAYMENT.COLLECT_ID	The collection ID	7
Act.Id	MSE_PAYMENT.ACTI_ID	The activity ID	8

The table key is “openItems”.

Header	MBO	Description	Table ID
Document Number	MSE_OPEN_ITEM.DOC_NO	Document number of the open item	0
Due Date	MSE_OPEN_ITEM.BLINE_DATE / DAYS_ARREARS	Due date of the open item (start date + days arrears)	1
Amount of Open Item	MSE_OPEN_ITEM.AMT_DOCCUR	Amount of the open item	2
Start Date	MSE_OPEN_ITEM.BLINE_DATE	Start date (BLINE) of the open item	3
Co.Code	MSE_OPEN_ITEM.COMP_CODE	Company code	4
Cust. No.	MSE_OPEN_ITEM.CUST_NO	Customer number	5
Day Arr.	MSE_OPEN_ITEM.DAYS_ARREARS	Days arrears for the open item	6
Dist.Chnl	MSE_OPEN_ITEM.VTWEG	Distribution channel	7
Div.	MSE_OPEN_ITEM.SPART	Division	8
Doc. Date	MSE_OPEN_ITEM.DOC_DATE	Document date	9
Doc.Type	MSE_OPEN_ITEM.DOC_TYPE	Document type	10
Fisc.Year	MSE_OPEN_ITEM.FISC_YEAR	Fiscal year	11
Client Created	MSE_OPEN_ITEM.HH_CREATE	Whether this open item is created on the client	12
Cancel	MSE_OPEN_ITEM.CANCELED	Whether this open item is canceled	13
Item Text	MSE_OPEN_ITEM.ITEM_TEXT	Item text	14
LineItemNo	MSE_OPEN_ITEM.ITEM_NUM	Line item number	15
RefDocNo	MSE_OPEN_ITEM.REF_DOC_NO	Reference document number	16
SalesOrg	MSE_OPEN_ITEM.VKORG	Sales organization	17

4.4.10 Invoice Credit Note Report Data

The template file is “Invoice_Report.txt” or “Invoice_Report_With_Empties_Section.txt”, if empties management is enabled. Also, there are other key fields available for this report (see 4.4.8 Partner Report Data). Additionally, this template supports signatures for the “Customer” and “Driver” roles.

4.4.10.1 Key Fields

Key	MBO	Description
invoiceText	MSE_VISIT_ACTIVITY.ACTI_ID	Localized text dependent on whether the report is an invoice or a credit note
invoiceNumber	MSE_INVOICE_HEADER.OFFICIAL_INV_NO	Official invoice number
creditNoteNumber	MSE_INVOICE_HEADER.OFFICIAL_INV_NO	Official invoice number
invoiceDate	MSE_INVOICE_HEADER.INV_DATE	Invoice date
creditNoteDate	MSE_INVOICE_HEADER.INV_DATE	Invoice date
paymentTerms	MSE_CUSTOMIZING “TERMS_OF_PAYMT” *Payment Terms Code*	Terms of payment
dueDate	-	Invoice date + days of payment terms
totalPrice	-	Sum of net price and tax of all invoice items
totalTax	MSE_INVOICE_ITEM.TAX	Sum of all taxes of all invoice items
totalExtended	MSE_INVOICE_ITEM.NET_PRICE	Sum of all net prices of all invoice items
CustomerNumber	MSE_EMPTIES_BALANCE.MATNR	Customer number
SalesOrganization	MSE_EMPTIES_BALANCE.VKORG	Sales organization
DistributionChannel	MSE_EMPTIES_BALANCE.VTWEG	Distribution channel
Division	MSE_EMPTIES_BALANCE.SPART	Division
currency	MSE_CUSTOMIZING “CURRENCY” – “EUR”	The customized currency for this shipment
dealConditionsBlock	-	Block element, see block section below

4.4.10.2 Table Fields

The table key is “materials”.

Header	MBO	Description	Table ID
Material	MSE_INVOICE_ITEM.MAT_NO	Formatted material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material Description	1
UoM	MSE_INVOICE_ITEM.DACT_UOM	Invoice item’s actual UOM	2
Qty	MSE_INVOICE_ITEM.DACT_QTY	Invoice item’s quantity	3
Price	MSE_INVOICE_ITEM.NET_PRICE	Invoice item’s net price divided by the actual quantity (net / actual qty)	4
TotNet	MSE_INVOICE_ITEM.NET_PRICE	Invoice item’s net price	5
Tax1 {Actual Tax Description}	MSE_DELIVERY_HEADER.TAX1_VALUE	Formatted tax value	6
Tax2 {Actual Tax Description}	MSE_DELIVERY_HEADER.TAX2_VALUE	Formatted tax value	7
Tax3 {Actual Tax Description}	MSE_DELIVERY_HEADER.TAX3_VALUE	Formatted tax value	8
Tax4 {Actual Tax Description}	MSE_DELIVERY_HEADER.TAX4_VALUE	Formatted tax value	9
Tax	MSE_INVOICE_ITEM.TAX	Invoice item tax	10
Total	-	Invoice item net price + invoice item tax	11
UePos	MSE_INVOICE_ITEM.INV_UEPOS		12
DocNo	MSE_INVOICE_ITEM.INV_NO	Invoice number	13
DocItemNo	MSE_INVOICE_ITEM.INV_ITM_NO	Invoice item number	14
DlvNo	MSE_INVOICE_ITEM.DELV_NO	Delivery number	15
DlvItemNo	MSE_INVOICE_ITEM.DELV_ITM_NO	Delivery item number	16
ImpEmpty	MSE_INVOICE_ITEM.IMP_EMPTY	Flag whether this item is an implied empty	17
Unt.Emp	MSE_INVOICE_ITEM.UNTIED_EMPTY	Flag whether this item is an untied empty	18
Mod	MSE_INVOICE_ITEM.MOD	Flag whether this item has been modified	19

Header	MBO	Description	Table ID
Promo.	MSE_INVOICE_ITEM.PROMO_RESULT	Flag whether this item is a promotion result	20
Return	MSE_INVOICE_ITEM.RTN_FLAG	Flag whether this item is a return	21
Billable	MSE_INVOICE_ITEM.BILLING_FLAG	Flag whether this item is billable	22
PromoNo	MSE_INVOICE_ITEM.PROMO_NO	The promotion number of the invoice item	23

The table key is “materialSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all net values. The second row shows the sum of all taxes and the third row show the sum of net values and taxes.

Header	MBO	Description	Table ID
1	-	The label for this row (Total Net, Total Tax, Total Due)	0
2	-	The actual value (sum)	1

The table key is “empties”. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
Empties	MSE_INVOICE_ITEM.MAT_NO	Formatted material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1
Initial	MSE_EMPTIES_BALANCE.END_BALANCE	The initial value of the item (end balance – actual balance)	2
Deliv.	MSE_INVOICE_ITEM.BILL_QTY	Number of delivered items	3
Return.	MSE_INVOICE_ITEM.BILL_QTY	Number of returned items	4
Actual	-	Actual number of items (delivered – returned)	5
Final	-	Final number of items (initial + actual)	6
Price	MSE_INVOICE_ITEM.NET_PRICE	Net price of the item	7
Tax	MSE_INVOICE_ITEM.TAX	Invoice item tax	8
TotNet	-	Actual * price	9
Total	-	Total Net + Tax	10

The table key is “emptiesSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all net values. The second row shows the sum of all taxes and the third row show the sum of net values and taxes. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
1	-	The label for this row (Empties Net, Empties Tax, Empties Due)	0
2	-	The actual value (sum)	1

The table key is “overallSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all invoice items (including taxes). The second row shows the sum of all empties and the third row show the overall sum. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
1	-	The label for this row (Total Due, Total Empties, Overall Due)	0
2	-	The actual value (sum)	1

4.4.10.3 Block Fields

The template for this block report is “Invoice_Report_DealConditions_Block.txt”. This block is created for each deal condition that has invoice items as free good result.

4.4.10.3.1 Key Fields

Key	MBO	Description
groupTitle	-	Translated label for deal condition number
groupDescription	-	Translated label for deal condition description
promoNumber	MSE_INVOICE_ITEM.PROMO_NO	The deal condition number
description	MSE_DEALCOND_HEAD.TEXT	The description of the deal condition

4.4.10.3.2 Table Fields

The table key is “dealConditionsTable”.

Header	MBO	Description	Table ID
Material	MSE_INVOICE_ITEM.MAT_NO	Formatted material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1
UoM	MSE_INVOICE_ITEM.DACT_UOM	Invoice item’s actual UOM	2
Qty	MSE_INVOICE_ITEM.DACT_QTY	Invoice item’s quantity	3
Reason	MSE_INVOICE_ITEM.INV_CHG_REAS	Change reason of the invoice item	4

4.4.11 Order Report Data

The template file is “Order_Report.txt” or “Order_Report_With_Empties_Section.txt”, if empties management is enabled. Also, there are other key fields available for this report (see 4.4.8 Partner Report Data). Additionally, this template supports signatures for the “Customer” and “Driver” roles.

4.4.11.1 Key Fields

Key	MBO	Description
deliveryDate	MSE_ORDER_HEADER.DELV_DATE	Date of the delivery
dueDate	-	Date of the delivery + days of payment terms
orderDate	MSE_ORDER_HEADER.DAT	Date of the order
orderNumber	MSE_ORDER_HEADER.OFFICIAL_ORD_NO / ORD_NO	The official order number or, if empty, the order number
orderType	MSE_VISIT_ACTIVITY.ACTI_TYPE	Localized type of the activity (order, free goods, return)
poNumber	MSE_ORDER_HEADER.PO_NO	Purchase order number
terms	MSE_CUSTOMIZING “TERMS_OF_PAYMT” *Payment Terms Code*	Terms of payment

Key	MBO	Description
total	MSE_ORDER_ITEM.NET_PRICE / TAX	Sum of all net prices and taxes overall order items
currency	MSE_CUSTOMIZING “CURRENCY” – “EUR”	The customized currency for this shipment
dealConditionsBlock	-	Block element, see block section below

4.4.11.2 Table Fields

The table key is “orderItems”.

Header	MBO	Description	Table ID
Material	MSE_ORDER_ITEM.MAT_NO	Material number	0
Description	MSE_ORDER_ITEM.MAT_DESC1	Material description	1
Qty	MSE_ORDER_ITEM.BILL_QTY	Billing quantity	2
UoM	MSE_ORDER_ITEM.SACT_UOM	Unit of measure	3
Price	-	Order item price (net price / billing quantity)	4
Ext.	MSE_ORDER_ITEM.NET_PRICE	Item’s net price	5
Tax	MSE_ORDER_ITEM.TAX	Item’s tax	6
Total	-	Order item’s net price + tax	7

The table key is “materialSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all net values. The second row shows the sum of all taxes and the third row show the sum of net values and taxes.

Header	MBO	Description	Table ID
1	-	The label for this row (total net, total tax, total due)	0
2	-	The actual value (sum)	1

The table key is “empties”. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
Empties	MSE_ORDER_ITEM.MAT_NO	Formatted material number	0

Header	MBO	Description	Table ID
Description	MSE_ORDER_ITEM.MAT_DESC 1	Material description	1
Initial	MSE_EMPTIES_BALANCE.END _BALANCE	The initial value of the item (end balance – actual balance)	2
Deliv.	MSE_ORDER_ITEM.BILL_QTY	Number of delivered items	3
Return.	MSE_ORDER_ITEM.BILL_QTY	Number of returned items	4
Actual	-	Actual number of items (delivered – returned)	5
Final	-	Final number of items (initial + actual)	6
Price	MSE_ORDER_ITEM.NET_PRIC E	Net price of the item	7
Total	-	Total net + tax	8

The table key is “emptiesSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all net values. The second row shows the sum of all taxes and the third row show the sum of net values and taxes. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
1	-	The label for this row (empties net, empties tax, empties due)	0
2	-	The actual value (sum)	1

The table key is “overallSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all order items (including taxes). The second row shows the sum of all empties and the third row show the overall sum. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
1	-	The label for this row (total due, total empties, overall due)	0
2	-	The actual value (sum)	1

4.4.11.3 Block Fields

Template for this block report is “Order_Report_DealConditions_Block.txt”. This block is created for each deal condition that has order items as free good result.

4.4.11.3.1 Key Fields

Key	MBO	Description
groupTitle	-	Translated label for deal condition number
groupDescription	-	Translated label for deal condition description
promoNumber	MSE_INVOICE_ITEM.PROMO_NO	The deal condition number
description	MSE_DEALCOND_HEAD.TEXT	The description of the deal condition

4.4.11.3.2 Table Fields

The table key is “dealConditionsTable”.

Header	MBO	Description	Table ID
Material	MSE_ORDER_ITEM.MAT_NO	Formatted material number	0
Description	MSE_ORDER_HD.MAT_DESC1	Material description	1
UoM	MSE_ORDER_ITEM.SACT_UOM	Order item’s actual UOM	2
Qty	MSE_ORDER_ITEM.SACT_UOM	Order item’s quantity	3
Reason	MSE_ORDER_ITEM.CHG_REAS	Change reason of the order item	4

4.4.12 Visit Pick List Report Data

The template file is “VisitPickList_Report.txt”. Also, there are other key fields available for this report (see 4.4.8 Partner Report Data).

4.4.12.1 Key Fields

Key	MBO	Description
totalMaterialQty	MSE_DELIVERY_ITEM.DACT_QTY	Sum of all delivery item quantities

4.4.12.2 Table Fields

The table key is “materials”.

Header	MBO	Description	Table ID
PO No.	MSE_DELIVERY_HEADER.PO_NO	Purchase order number	0
Delivery Date	MSE_DELIVERY_HEADER.DAT	Delivery date	1
Material	MSE_DELIVERY_ITEM.MAT_NO	Material number	2
Material Name	MSE_DELIVERY_ITEM.MAT_DESC1	Material description	3
UoM	MSE_DELIVERY_ITEM.DACT_UOM	Material UoM	4
Quantity.	MSE_DELIVERY_ITEM.DACT_QTY	Actual quantity of material	5
Return	MSE_DELIVERY_ITEM.RTN_FLAG	Flag if delivery item is a return	6

4.4.13 Delivery Note Report Data

The template file is “DeliveryNote_Report.txt” or “DeliveryNote_Report_With_Empties_Section.txt”, if empties management is enabled. Also, there are other key fields available for this report (see 4.4.8 Partner Report Data). Additionally, this template supports signatures for the “Customer” and “Driver” roles.

4.4.13.1 Key Fields

Key	MBO	Description
DeliveryNote	MSE_VISIT_ACTIVITY.ACTI_TYPE	Localized type of the activity (delivery, free goods, return)
DeliveryNumber	MSE_DELIVERY_HEADER.DELV_NO	Official delivery number
DeliveryDate	MSE_DELIVERY_HEADER.DELV_DATE	Delivery date
CustomerNumber	MSE_EMPTIES_BALANCE.MATNR	Customer number
SalesOrganization	MSE_EMPTIES_BALANCE.VKORG	Sales organization
DistributionChannel	MSE_EMPTIES_BALANCE.VTWEG	Distribution channel

Key	MBO	Description
Division	MSE_EMPTIES_BALANCE.SPART	Division
ReferenceNumber	MSE_DELIVERY_HEADER.PO_NO	Purchase order number
PaymentTerms	MSE_CUSTOMIZING "TERMS_OF_PAYMT" *Payment Terms Code*	Terms of payment
DueDate	-	Due date (delivery date + days of payment terms)
dealConditionsBlock	-	Block element, see block section below

4.4.13.2 Table Fields

The table key is "materials".

Header	MBO	Description	Table ID
Material	MSE_DELIVERY_ITEM.MAT_NO	Formatted material number	0
Description	MSE_DELIVERY_ITEM.MAT_DESC1	Material description	1
Quantity	MSE_DELIVERY_ITEM.DACT_QTY	Delivery item's quantity	2
UoM	MSE_DELIVERY_ITEM.DACT_UOM	Invoice item's actual UOM	3
Reason	MSE_DELIVERY_ITEM.CHG_REAS / MSE_CUSTOMIZING	Change reason of delivery item	4
Bil.Qty	MSE_DELIVERY_ITEM.BILL_QTY	Billing quantity	5
Billable	MSE_DELIVERY_ITEM.BILLING_FLAG	Flag whether this item is billable	6
NetPrice	MSE_DELIVERY_ITEM.NET_PRICE	Delivery item's net price	7
Tax	MSE_DELIVERY_ITEM.TAX	Tax of delivery item	8
TaxRate	MSE_DELIVERY_ITEM.TAX_RATE	Tax rate of delivery item	9
UnEmpty	MSE_DELIVERY_ITEM.UNTIED_EMPTY	Flag whether this item is an untied empty	10
PromoNo	MSE_DELIVERY_ITEM.PROMO_NO	The promotion number of the delivery item	11
PromoRes	MSE_DELIVERY_ITEM.PROMO_RESULT	Flag whether this item is a promotion result	12

Header	MBO	Description	Table ID
Importance	-	Importance of the item	13
UePos	MSE_DELIVERY_ITEM.INV_UEPOS		14

The table key is “empties”. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
Empties	MSE_DELIVERY_ITEM.MAT_NO	Formatted material number	0
Description	MSE_DELIVERY_ITEM-MAT_DESC1	Material description	1
Initial	MSE_EMPTIES_BALANCE.END_BALANCE	The initial value of the item (end balance – actual balance)	2
Deliv.	MSE_DELIVERY_ITEM.BILL_QTY	Number of delivered items	3
Return.	MSE_DELIVERY_ITEM.BILL_QTY	Number of returned items	4
Actual	-	Actual number of items (delivered – returned)	5
Final	-	Final number of items (initial + actual)	6

4.4.13.3 Block Fields

Template for this block report is “Delivery_Report_DealConditions_Block.txt”. This block is created for each deal condition that has invoice items as free good result.

4.4.13.3.1 Key Fields

Key	MBO	Description
groupTitle	-	Translated label for deal condition number
groupDescription	-	Translated label for deal condition description
promoNumber	MSE_DELIVERY_ITEM.PROMO_NO	The deal condition number
description	MSE_DEALCOND_HEAD.TEXT	The description of the deal condition

4.4.13.3.2 Table Fields

The table key is “dealConditionsTable”.

Header	MBO	Description	Table ID
Material	MSE_DELIVERY_ITEM.MAT_NO	Formatted material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1
UoM	MSE_DELIVERY_ITEM.DACT_UOM	Delivery item’s actual UOM	2
Qty	MSE_DELIVERY_ITEM.DACT_QTY	Delivery item’s quantity	3
Reason	MSE_DELIVERY_ITEM.CHG_REAS	Change reason of the delivery item	4

4.4.14 Valuated Delivery Note Report Data

Template file is “DeliveryNote_Report_With_Empties_Section.txt”. Also, there are other key fields available for this report (see 4.4.8 Partner Report Data). Additionally, this template supports signatures for the “Customer” and “Driver” roles.

4.4.14.1 Key Fields

Key	MBO	Description
deliveryText	-	Localized text for this report
deliveryNumber	MSE_DELIVERY_HEADER.DELV_NO	Official delivery number
deliveryDate	MSE_DELIVERY_HEADER.DAT	Date
paymentTerms	MSE_CUSTOMIZING “TERMS_OF_PAYMT” *Payment Terms Code*	Terms of payment
dueDate	-	Invoice date + days of payment terms
totalPrice	-	Sum of net price and tax of all invoice items
totalTax	MSE_INVOICE_ITEM.TAX	Sum of all taxes of all invoice items
totalExtended	MSE_INVOICE_ITEM.NET_PRICE	Sum of all net prices of all invoice items
CustomerNumber	MSE_EMPTIES_BALANCE.MATNR	Customer number

Key	MBO	Description
SalesOrganization	MSE_EMPTIES_BALANCE.VKORG	Sales organization
DistributionChannel	MSE_EMPTIES_BALANCE.VTWEG	Distribution channel
Division	MSE_EMPTIES_BALANCE.SPART	Division
currency	MSE_CUSTOMIZING “CURRENCY” – “EUR”	The customized currency for this shipment
dealConditionsBlock	-	Block element, see block section below

4.4.14.2 Table Fields

The table key is “materials”.

Header	MBO	Description	Table ID
Material	MSE_INVOICE_ITEM.MAT_NO	Formatted material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1
UoM	MSE_INVOICE_ITEM.DACT_UOM	Invoice item’s actual UOM	2
Qty	MSE_INVOICE_ITEM.DACT_QTY	Invoice item’s quantity	3
Price	MSE_INVOICE_ITEM.NET_PRICE	Invoice item’s net price divided by the actual quantity (net / actual qty)	4
TotNet	MSE_INVOICE_ITEM.NET_PRICE	Invoice item’s net price	5
Tax1 {Actual Tax Description}	MSE_DELIVERY_HEADER.TAX1_VALUE	Formatted tax value	6
Tax2 {Actual Tax Description}	MSE_DELIVERY_HEADER.TAX2_VALUE	Formatted tax value	7
Tax3 {Actual Tax Description}	MSE_DELIVERY_HEADER.TAX3_VALUE	Formatted tax value	8
Tax4 {Actual Tax Description}	MSE_DELIVERY_HEADER.TAX4_VALUE	Formatted tax value	9
Tax	MSE_INVOICE_ITEM.TAX	Invoice item tax	10
Total	-	Invoice item net price + invoice item tax	11
UePos	MSE_INVOICE_ITEM.INV_UEPOS		12

Header	MBO	Description	Table ID
DocNo	MSE_INVOICE_ITEM.INV_NO	Invoice number	13
DocItemNo	MSE_INVOICE_ITEM.INV_ITM_NO	Invoice item number	14
DlvNo	MSE_INVOICE_ITEM.DELV_NO	Delivery number	15
DlvItemNo	MSE_INVOICE_ITEM.DELV_ITM_NO	Delivery item number	16
ImpEmpty	MSE_INVOICE_ITEM.IMP_EMPTY	Flag whether this item is an implied empty	17
Unt.Emp	MSE_INVOICE_ITEM.UNTIED_EMPTY	Flag whether this item is an untied empty	18
Mod	MSE_INVOICE_ITEM.MOD	Flag whether this item has been modified	19
Promo.	MSE_INVOICE_ITEM.PROMO_RESULT	Flag whether this item is a promotion result	20
Return	MSE_INVOICE_ITEM.RTN_FLAG	Flag whether this item is a return	21
Billable	MSE_INVOICE_ITEM.BILLING_FLAG	Flag whether this item is billable	22
PromoNo	MSE_INVOICE_ITEM.PROMO_NO	The promotion number of the invoice item	23

The table key is “materialSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all net values. The second row shows the sum of all taxes and the third row show the sum of net values and taxes.

Header	MBO	Description	Table ID
1	-	The label for this row (total net, total tax, total due)	0
2	-	The actual value (sum)	1

The table key is “empties”. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
Empties	MSE_INVOICE_ITEM.MAT_NO	Formatted material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1
Initial	MSE_EMPTIES_BALANCE.END_BALANCE	The initial value of the item (end balance – actual balance)	2
Deliv.	MSE_INVOICE_ITEM.BILL_QTY	Number of delivered items	3
Return.	MSE_INVOICE_ITEM.BILL_QTY	Number of returned items	4

Header	MBO	Description	Table ID
Actual	-	Actual number of items (delivered – returned)	5
Final	-	Final number of items (initial + actual)	6
Price	MSE_INVOICE_ITEM.NET_PRICE	Net price of the item	7
Tax	MSE_INVOICE_ITEM.TAX	Invoice item tax	8
TotNet	-	Actual * price	9
Total	-	Total Net + Tax	10

The table key is “emptiesSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all net values. The second row shows the sum of all taxes and the third row show the sum of net values and taxes. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
1	-	The label for this row (empties net, empties tax, empties due)	0
2	-	The actual value (sum)	1

The table key is “overallSum”. This table is used to print the sum of all items in a formatted way. The first row shows the sum of all invoice items (including taxes). The second row shows the sum of all empties and the third row show the overall sum. This table is only available if empties management is enabled.

Header	MBO	Description	Table ID
1	-	The label for this row (total due, total empties, overall due)	0
2	-	The actual value (sum)	1

4.4.14.3 Block Fields

The template for this block report is “Invoice_Report_DealConditions_Block.txt”. This block is created for each deal condition that has invoice items as free good result.

4.4.14.3.1 Key Fields

Key	MBO	Description
groupTitle	-	Translated label for deal condition number
groupDescription	-	Translated label for deal condition description
promoNumber	MSE_INVOICE_ITEM.PROMO_NO	The deal condition number
description	MSE_DEALCOND_HEAD.TEXT	The description of the deal condition

4.4.14.3.2 Table Fields

The table key is “dealConditionsTable”.

Header	MBO	Description	Table ID
Material	MSE_INVOICE_ITEM.MAT_NO	Formatted material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1
UoM	MSE_INVOICE_ITEM.DACT_UOM	Invoice item’s actual UOM	2
Qty	MSE_INVOICE_ITEM.DACT_QTY	Invoice item’s quantity	3
Reason	MSE_INVOICE_ITEM.INV_CHG_REAS	Change reason of the invoice item	4

4.4.15 Pricing Information Data

The pricing information table is available within the invoice report (see Invoice Credit Note Report Data), the valuated delivery note report (see Valuated Delivery Note Report Data), and the order report (see Order Report Data).

4.4.15.1 Key Fields

None

4.4.15.2 Table Fields

The table key is “pricingResults”.

Header	MBO	Description	Table ID
DocumentNumber	MSE_PRICING_COND_RESULT .DOCUMENT_NO	Related document number	0
ItemNumber	MSE_PRICING_COND_RESULT .	Document item number	1
Handle	MSE_PRICING_COND_RESULT .HANDLE	GUID	2
MaterialNumber	MSE_PRICING_COND_RESULT .MAT_NO	Material number of the document item	3
ConditionType	MSE_PRICING_COND_RESULT .KSCHL	Condition type	4
Counter	MSE_PRICING_COND_RESULT .COUNTER	Counter	5
RateInternal	MSE_PRICING_COND_RESULT .KBETR_INT	Rate (condition amount or percentage)	6
RateExternal	MSE_PRICING_COND_RESULT .KBETR_EXT	Rate (condition amount or percentage)	7
RateUnit	MSE_PRICING_COND_RESULT .KOEIN	Rate unit (currency, sales unit, or %)	8
RateUnitInternal	MSE_PRICING_COND_RESULT .KOEIN_INT	Rate unit (currency, sales unit, or %)	9
ConditionPricingUnit	MSE_PRICING_COND_RESULT .KPEIN	Condition pricing unit	10
ConditionUnitInDoc	MSE_PRICING_COND_RESULT .KMEIN	Condition unit in the document	11
ConditionValueInternal	MSE_PRICING_COND_RESULT .KWERT_INT	Internal condition value	12
ConditionValueExternal	MSE_PRICING_COND_RESULT .KWERT_EXT	External condition value	13
DocumentCurrency	MSE_PRICING_COND_RESULT .WAERK	Document currency	14
ConversionNumerator	MSE_PRICING_COND_RESULT .KUMZA	Numerator for converting condition units to base units	15
ConversionDenominator	MSE_PRICING_COND_RESULT .KUMNE	Denominator for converting condition units to base units	16
UnitOfMeasureBase	MSE_PRICING_COND_RESULT .MEINS	Base UoM	17
ConditionUnit	MSE_PRICING_COND_RESULT .KMEI1	Condition unit	18

Header	MBO	Description	Table ID
IsInactive	MSE_PRICING_COND_RESULT .KINAK	Flag whether condition is inactive	19
SelectionFlag	MSE_PRICING_COND_RESULT .SELKZ	Selection flag	20
ConditionClass	MSE_PRICING_COND_RESULT .KOAID	Condition class (A – Discount/Surcharge, B – Prices, C – Expense/Reimbursement, D – Tax)	21
CalculationType	MSE_PRICING_COND_RESULT .KRECH	Calculation Type (A – Percentage, B – Fixed amount, C – Quantity, D – Gross weight, E – Net weight, F – Volume, G – Formula, H – Percentage included)	22

4.4.16 Visit List Overview Report Data

The template file is “Visit_List_Overview_Report.txt”.

4.4.16.1 Key Fields

None

4.4.16.2 Table Fields

The table key is “activities”.

Header	MBO	Description	Table ID
Visit	-	Index of the visit (incremented by each visit)	0
Accounts	MSE_VISIT_HEADER.CUST_NO	Customer number of the visit	1
Name/Activity	MSE_VISIT_HEADER.NAME MSE_VISTIT_ACTIVITY.ACTI_ID	Customer name of visit Localized name of the activity type	2
Start/End	MSE_VISIT_HEADER.PSTART_DATE / PEND_DATE MSE_VISIT_ACTIVITY.BEG_DATE / END_DATE	Start/End date of the visit Start/End date of activity	3

4.4.17 Visit List Details Report Data

The template file is “Visit_List_Details_Report.txt”.

4.4.17.1 Key Fields

Key	MBO	Description
VisitCount		Number of visits with activities
ActivitiesCount		Number of all activities
TotalTime		Total time of all activities
Visits	-	Block key, see block section below 4.4.17.3 Block Fields.

4.4.17.2 Table Fields

None

4.4.17.3 Block Fields

The template for this block report is “Visit_List_Details_Report_VisitTable.txt”. This block is created for each visit. Also, this block contains another block, see section below.

4.4.17.3.1 Key Fields

Key	MBO	Description
activities	-	Block key, see block section below 4.4.17.4 Block Fields.

4.4.17.3.2 Table Fields

The table key is “visit”.

Header	MBO	Description	Table ID
Visit	-	Index of the visit (incremented by each visit)	0
Accounts	MSE_VISIT_HEADER.CUST_NO	Customer number	1
Name/Activity	MSE_VISIT_HEADER.NAME	Customer name of visit	2
Start/End	MSE_VISIT_HEADER.PSTART_DATE / PEND_DATE	Start/End date of the visit	3

4.4.17.4 Block Fields

Template for this block report is “Visit_List_Details_Report_VisitTable_MaterialTable.txt”. This block is created for each activity within a visit.

4.4.17.4.1 Key Fields

Key	MBO	Description
Activity	MSE_VISTI_ACTIVITY.ACTI_ID	Localized text of the current activity type
Total	MSE_DELIVERY_ITEM.DACT_QTY / MSE_ORDER_ITEM.DACT_QTY	Sum of all actual quantities of an item within an order or delivery
TotalTime	MSE_VISIT_ACTIVITY.END_TIME	The end time of the current activity

4.4.17.4.2 Table Fields

The table key is “materials”.

Header	MBO	Description	Table ID
Material	MSE_DELIVERY_ITEM.MAT_NO / MSE_ORDER_ITEM.MAT_NO	Material number	0
Material Name	MSE_DELIVERY_ITEM.MAT_DESC1 / MSE_ORDER_ITEM.MAT_DESC1	Description of the material	1

Header	MBO	Description	Table ID
UoM	MSE_DELIVERY_ITEM.DACT_UOM / MSE_ORDER_ITEM.DACT_UOM	Unit of measure of the material	2
Quantity	MSE_DELIVERY_ITEM.DACT_QTY / MSE_ORDER_ITEM.DACT_QTY	Quantity of the material	3

4.4.18 Visit List Discrepancies Report Data

The template file is “Visit_List_Discrepancies_Report.txt”.

4.4.18.1 Key Fields

None

4.4.18.2 Table Fields

The table key is “visit”.

Header	MBO	Description	Table ID
Visit	-	Index of the visit (incremented by each visit)	0
Accounts	MSE_VISIT_HEADER.CUST_NO	Customer number	1
Name/Activity	MSE_VISIT_HEADER.NAME	Customer name of visit	2
Plan	MSE_VISIT_HEADER.PSTART_DATE / PEND_DATE	Start date/End date of the visit	3
Actual	MSE_VISIT_ACTIVITY.BEG_DATE / END_DATE	Actual start/end of the visit based on the activities	4
Deviation	-	Deviation of planned start/end to actual start/end	5

4.4.19 Route Performance Report Data

The template file is “Route_Performance.txt”.

4.4.19.1 Key Fields

Key	MBO	Description
PlannedVisits	-	Count of all visits
ExecutedVisits	-	Count of executed visits
EffectiveVisits	-	Count of successful visits
VisitEffectiveness	-	Effectiveness percentage (effective visits/planned visits)
SalesEffectiveness	-	Sales effectiveness percentage (effective visits/planned visits)
VisitsWithoutSalesReason	-	Number of visits that have cancelled documents in it (in percentage)
VisitsWithoutVisitReason	MSE_VISIT_HEADER.ACT_STAT_REAS	Number of visits without a visit status reason (in percentage)
CustomerWithoutSalesOrVisitReason	-	Sum of visits without sales or visit reason

4.4.19.2 Table Fields

The table key is “materials”.

Header	MBO	Description	Table ID
Material	MSE_MATERIAL_HD.MAT_NO	Material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1
Sales SUOM/BUOM	-	Material quantity in sales UOM and base UOM	2
Returns SUOM/BUOM	-	Material return quantity in sales UOM and base UOM	3
Free Goods SUOM/BUOM	-	Material free goods quantity in sales UOM and base UOM	4

The table key is “cancelled materials”.

Header	MBO	Description	Table ID
Material	MSE_MATERIAL_HD.MAT_NO	Material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1

Header	MBO	Description	Table ID
Sales SUOM/BUOM	-	Material quantity in sales UOM and base UOM	2
Returns SUOM/BUOM	-	Material return quantity in sales UOM and base UOM	3
Free Goods SUOM/BUOM	-	Material free goods quantity in sales UOM and base UOM	4

The table key is “empties”.

Header	MBO	Description	Table ID
Empties	MSE_MATERIAL_HD.MAT_NO	Material number	0
Description	MSE_MATERIAL_HD.MAT_DESC1 / MAT_DESC2	Material description	1
Returns	-	Empties return quantity in base UOM	2

The table key is “salesReasons”.

Header	MBO	Description	Table ID
Sales Reason Code	MSE_DELIVERY_HEADER.CAN_REAS MSE_INVOICE_HEADER.CAN_REAS MSE_ORDER_HEADER.CAN_REAS	Cancel reason code of document (delivery, invoice, or order)	0
Number of Customers	-	Count customers with cancel reason	1

The table key is “visitReasons”.

Header	MBO	Description	Table ID
Visit Reason Code	MSE_VISIT_HEADER.ACT_STAT_REAS	Visit reason code	0
Number of Customers	MSE_VISIT_HEADER.CUST_NO	Customer number	1

4.4.20 Presettlement Report Data

The template file is “Presettlement_Report.txt”.

4.4.20.1 Key Fields

Key	MBO	Description
TotalCollected	-	Amount of all collections
FirstDocumentNumber	MSE_COLLECTION.COLLECT_ID	First taken collection
LastDocumentNumber	MSE_COLLECTION.COLLECT_ID	Last taken collection
NumberOfDpcuments	-	Number of collections taken
NumberOfCanceledDocuments	-	Number of cancelled collections
TotalCheck	-	Amount of all check payments
TotalCreditCard	-	Amount of all credit card payments
TotalCustom	-	Amount of all custom payments
TotalBank	-	Amount of all bank payments

4.4.20.2 Table Fields

The table key is “paymentsList”.

Header	MBO	Description	Table ID
Payment Method	MSE_PAYMENT.PYMT_TYPE MSE_CUSTOMIZING – “PYMT_TYPE”	Localized payment method	0
Total Payments	-	Number of payments taken for this payment method	1
Total Amount	-	Total amount taken for this payment method	2

The table key is “cancelledList”.

Header	MBO	Description	Table ID
{Empty}	-	Empty column	0
DocumentNumber	MSE_COLLECTION.COLELCT_ID	Document number of the canceled collection	1

The table key is “checkList”.

Header	MBO	Description	Table ID
Check Number	MSE_PAYMENT.CHECK_NO	The check number	0
Bank Name	MSE_PAYMENT.BK_KEY	The bank name	1
Amount Collected	MSE_PAYMENT.AMOUNT	Amount of payment	2

The table key is “creditList”.

Header	MBO	Description	Table ID
Credit Card Number	MSE_PAYMENT.PY_C_NUM	Credit card number	0
Bank Name	MSE_PAYMENT.PY_C_TYPE	Credit card type	1
Amount Collected	MSE_PAYMENT.AMOUNT	Amount of payment	2

The table key is “customList”.

Header	MBO	Description	Table ID
ID	MSE_PAYMENT.CASH_ID	ID of custom payment	0
Method Name	MSE_PAYMENT.PYMT_TYPE MSE_CUSTOMIZING – “PYMT_TYPE”	Localized payment type	1
Amount Collected	MSE_PAYMENT.AMOUNT	Amount of payment	2

The table key is “bankList”.

Header	MBO	Description	Table ID
Deposit Number	MSE_PAYMENT.ACC_KEY		0
Bank Name	MSE_PAYMENT.BK_KEY		1
Amount Collected	MSE_PAYMENT.AMOUNT	Amount of payment	2

4.4.21 Inventory Report Data

The template file is “Empties_Balance_Report.txt”.

4.4.21.1 Key Fields

Key	MBO	Description
CustomerNumber	MSE_CUSTOMER_SA.CUST_NO	Customer number

Key	MBO	Description
CustomerName	MSE_CUSTOMER_HD.NAME	Customer name
CustomerStreet	MSE_CUSTOMER_HD.STREET	Customer street
CustomerCityAndPCode	MSE_CUSTOMER_HD.CITY / PCODE	Customer city and postal code
VatregNo	MSE_CUSTOMER_HD.VATREG_NO	Customer VAT region number
DriverCode	MSE_TOUR_HD.DRIVER_ID1	Driver name
TotalNet	MSE_DELVIERY_ITEM.NET_PRICE MSE_ORDER_ITEM.NET_PRICE	Sum of all net values
TotalTax	MSE_DELVIERY_ITEM.TAX MSE_ORDER_ITEM.TAX	Sum of all taxes
Total	-	Sum of TotalNet and TotalTax

4.4.21.2 Table Fields

The table key is “emptiesBalance”.

Header	MBO	Description	Table ID
Empty	MSE_DELVIERY_ITEM.MAT_NO MSE_ORDER_ITEM.MAT_NO	Material number	0
Description	MSE_DELVIERY_ITEM.MAT_DESC1 MSE_ORDER_ITEM.MAT_DESC1	Material description	1
Initial	MSE_EMPTIES_BALANCE.INIT_BALANCE	Initial amount	2
Deliv.	MSE_DELVIERY_ITEM.DACT_QTY MSE_ORDER_ITEM.BILL_QTY	Delivered amount	3
Return.	MSE_DELVIERY_ITEM.DACT_QTY MSE_ORDER_ITEM.BILL_QTY	Returned amount	4
Actual	-	Delivered - Returned	5
Final	-	Initial + Actual	6

Header	MBO	Description	Table ID
Price	MSE_DELVIERY_ITEM.NET_PRICE MSE_ORDER_ITEM.NET_PRICE	Net price of item	7
Total	-	Net price + tax of item	8

4.4.22 Expense Report Data

The template file is “Expense_Summary.txt”.

4.4.22.1 Key Fields

Key	MBO	Description
Total	MSE_EXPENSE.AMOUNT	Sum of expenses

4.4.22.2 Table Fields

The table key is “expenses”.

Header	MBO	Description	Table ID
Expense ID	MSE_EXPENSE.EXP_ID	Expense ID	0
Expense	MSE_EXPENSE.EXP_TYPE	Expense type	1
Price	MSE_EXPENSE.AMOUNT	Amount of the expense	2

4.4.23 Pending Collection Report

The template file is ‘PendingCollection_Report.txt’. This report supports signatures for the “Customer” and “Driver” roles.

4.4.23.1 Key Fields

None

4.4.23.2 Block Fields

The template for this block report is “PendingCollection_Report_Block.txt”. This block is created for each visit of the tour.

4.4.23.2.1 Key Fields

Key	MBO	Description
CustomerName	MSE_CUSTOMER_HD.NAME	Name of the customer
CustomerNumber	MSE_CUSTOMER_HD.CUST_NO	Number of the customer
SalesArea	MSE_CUSTOMER_SA.SALES_ORG / DIST_CHANNEL / DIVISION	Sales area of the customer (sales organization + distribution channel + division)
NumberOfInvoices	-	Count of all invoices
NumberOfDueItems	-	Count of all due items

4.4.23.2.2 Table Fields

The table key is “generatedDocumentsList”.

Header	MBO	Description	Table ID
Invoice Number	MSE_INVOICE_HEADER.OFFICIAL_INV_NO	Invoice number	0
Amount	MSE_INVOICE_HEADER.TOT_AMT	Total amount	1

The table key is “dueOpenItemsList”.

Header	MBO	Description	Table ID
Invoice Number	MSE_OPEN_ITEM.REF_DOC_NO	Invoice number	0
Amount	MSE_OPEN_ITEM.AMT_DOCCUR / MSE_CLEARING.ASS_AMOUNT	Remaining amount (Amount Document – sumo of Cleared amount)	1

Due Date	MSE_OPEN_ITEM.BLINE_DATE / DAYS_ARREARS	Due date of the open item (start date + days arrears)	2

Appendix

I. ExtensibilityPlugin Listing

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;

using com.sapcd.mobileclient.extensibilityplugin.Interfaces;
using com.sapcd.mobileclient.framework.Config;
using com.sap.afx.json;
using com.sapcd.mobileclient.framework.Logging;
using com.sapcd.mobileclient.dialog.WMForms.Controller;

using SAPCD.MobileSalesClient.Application.UserViewPlugin.About;

namespace MobileSalesExtensibilityTest
{
	class ExtensibilityPlugin : IExtensibilityPlugin
	{
		#region IExtensibilityPlugin Members

		private Dictionary<Type, Type> mapping;
		private Dictionary<string, Type> fallbackMapping;
		private List<Type> exceptions;
		private const string CONFIG_FILE = "\\extensibilityConfig.json";

		public ExtensibilityPlugin()
		{
			mapping = new Dictionary<Type, Type>();
			fallbackMapping = new Dictionary<string, Type>();
			exceptions = new List<Type>();

	LoadFromFile(ConfigProvider.Singleton.ConfigManager.ConfigDirectory +
CONFIG_FILE);
		}

		/// <summary>
```

```
        /// Gets the controller extended.
        /// </summary>
        /// <param name="controllerType">Type of the controller.</param>
        /// <returns></returns>
        public Type GetControllerExtended(Type controllerType)
        {
            Type choice;
            Type current = null;
            // check for first initialization of the client
            if (MobileMainDialogController.Singleton != null &&
MobileMainDialogController.Singleton.ActiveController != null)
            {
                current =
MobileMainDialogController.Singleton.ActiveController.GetType();
            }

            // Try finding type
            if (mapping.TryGetValue(controllerType, out choice))
            {
                // do nothing
            }
            else if(fallbackMapping.TryGetValue(controllerType.FullName, out
choice))
            {
                // for efficiency reasons add the Type to the original
mapping
                mapping.Add(controllerType, choice);
                fallbackMapping.Remove(controllerType.FullName);
            }
            else if(fallbackMapping.TryGetValue(controllerType.Name, out
choice))
            {
                // for efficiency reasons add the Type to the original
mapping
                mapping.Add(controllerType, choice);
                fallbackMapping.Remove(controllerType.Name);
            }
            else
            {
                // no rule found
                choice = controllerType;
            }
```

```
                // if the current controller is an exception, avoid self-loops
                if (exceptions.Contains(current) && choice == current)
                {
                        choice = controllerType;
                }
                return choice;
        }

        /// <summary>
        /// Gets the plugin id.
        /// </summary>
        public string PluginId
        {
                get { return "ExtensibilityPlugin.SAPCD.MSE.Application"; }
        }

        /// <summary>
        /// Loads from file.
        /// </summary>
        /// <param name="location">The location.</param>
        private void LoadFromFile(string location)
        {
                bool replace = false;
                string sourceEntry = string.Empty;
                string sourceDll = string.Empty;
                string targetEntry = string.Empty;
                string targetDll = string.Empty;

                Logger.Singleton.Log(LogLevel.Info, "Load file " + location,
"ExtensibilityPlugin");
                try
                {
                        // get the configuration as JSON formatted string
                        TextReader configReader = new StreamReader(location);
                        string configuration = configReader.ReadToEnd();
                        configReader.Close();
                        // parse the string to retrieve JSON object
                        JsonArray jsonConfigArray =
(JsonArray)JsonReader.Parse(configuration);
```

```
                        foreach (JsonObject configEntry in jsonConfigArray)
                        {
                                sourceEntry = configEntry.GetString("Source");
                                sourceDll = configEntry.GetString("SourceDllFile");
                                targetEntry = configEntry.GetString("Target");
                                targetDll = configEntry.GetString("TargetDllFile");
                                Type source = LoadType(sourceDll, sourceEntry);
                                Type target = LoadType(targetDll, targetEntry);
                                replace = configEntry.GetBoolean("Replace");
                                // only if the Replace-switch is "true"
                                if (replace)
                                {
                                        // check that types have been found
                                        if (target == null)
                                                continue;
                                        // use fallback mapping if neccessary
                                        if (source == null)
                                        {
                                                fallbackMapping.Add(sourceEntry,
target);
                                        }
                                        else
                                        {
                                                mapping.Add(source, target);
                                        }
                                        // we have to be able to call the source from
the target
                                        exceptions.Add(target);
                                }
                        }
                }
                catch (Exception e)
                {
                        Logger.Singleton.Log(LogLevel.Warn, "Error parsing
extensibility plugin's config file.", "ExtensibilityPlugin");
                        Logger.Singleton.Log(LogLevel.Warn, e.Message,
"ExtensibilityPlugin");
                        Logger.Singleton.Log(LogLevel.Warn, e.StackTrace,
"ExtensibilityPlugin");
                }
        }
```

```
            /// <summary>
            /// Loads the type.
            /// </summary>
            /// <param name="dllFileName">Name of the DLL file.</param>
            /// <param name="name">The name.</param>
            /// <returns></returns>
            private Type LoadType(string dllFileName, string name)
            {
                  // try loading the type from the name information
                  string n = string.Empty;
                  try
                  {
                        Assembly ass = Assembly.LoadFrom(dllFileName);
                        foreach (Type t in ass.GetTypes())
                        {
                              if(t.Name.Equals(name,
StringComparison.CurrentCultureIgnoreCase))
                              {
                                    return t;
                              }
                        }
                  }
                  catch (Exception ex)
                  {
                        Logger.Singleton.Log(ex);
                        Logger.Singleton.Log(LogLevel.Warn, string.Format("Error
parsing extensibility plugin's config file. Could not find type {0}. Please give full
name including assembly.", name), "ExtensibilityPlugin");
                  }
                  return null;
            }

            #endregion
      }
}
```

II. Extension JSON Configuration (Example)

```
    {
        "SourceDllFile": "SAPCD.MobileSalesClient.Application.dll",
          "Source":        "MseMainDialogController",
          "TargetDllFile": "MobileSalesClientExtensibilityTest.dll",
          "Target":        "MseMainDialogExtendedController",
          "Replace":       true
    },
    {
        "SourceDllFile": "SAPCD.MobileSalesClient.Application.dll",
          "Source":        "ReportPreviewController",
          "TargetDllFile": "MobileSalesClientExtensibilityTest.dll",
          "Target":        "PreReportsController",
          "Replace":       true
    },
    {
        "SourceDllFile": "SAPCD.MobileSalesClient.Application.dll",
          "Source":        "ReportPreviewController",
          "TargetDllFile": "MobileSalesClientExtensibilityTest.dll",
          "Target":        "PreReportsController",
          "Replace":       false
    }
]
```

III. AboutController Extension (Example)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using SAPCD.MobileSalesClient.Application.Reports.PrintPreview;
using SAPCD.MobileSalesClient.Application.UserViewPlugin.About;
using com.sapcd.mobileclient.dialog.Interfaces.Common;
using com.sapcd.mobileclient.dialog.Interfaces.Model;

namespace MobileSalesExtensibilityTest
{
	public class AboutViewExtendedController : AboutViewController
	{
		#region Overrides

		/// <summary>
		/// Gets the type of the view.
		/// </summary>
		/// <value>
		/// The type of the view.
		/// </value>
		public override Type ViewType
		{
			get
			{
				return typeof(AboutViewExtended);
			}
		}

		protected override void RegisterAction(Enum key,
com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroup ag)
		{
			ActionKeys tmpKey = (ActionKeys)key;
			switch (tmpKey)
			{
				case ActionKeys.Back:
				{
					ag.Action += BackActionHandler;
					break;
				}
```

```
                        case ActionKeys.Refresh:
                        {
                                ag.Action += RefreshActionHandler;
                                break;
                        }
                        case ActionKeys.SendLogs:
                        {
                                ag.Action += SendLogsActionHandler;
                                break;
                        }
                }

                base.RegisterAction(key, ag);
        }


        protected override void DeregisterAction(Enum key,
com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroup ag)
        {
                ActionKeys tmpKey = (ActionKeys)key;
                switch (tmpKey)
                {
                        case ActionKeys.Back:
                        {
                                ag.Action -= BackActionHandler;
                                break;
                        }
                        case ActionKeys.Refresh:
                        {
                                ag.Action -= RefreshActionHandler;
                                break;
                        }
                        case ActionKeys.SendLogs:
                        {
                                ag.Action -= SendLogsActionHandler;
                                break;
                        }
                }

                base.DeregisterAction(key, ag);
        }
```

```
            #endregion // Overrides

            #region ActionHandlers

            /// <summary>
            /// Syncs the back action handler.
            /// </summary>
            /// <param name="sender">The sender.</param>
            /// <param name="args">The <see
cref="com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroupEventArgs"/>
instance containing the event data.</param>
            private void BackActionHandler(IActionGroup sender, IActionGroupEventArgs
args)
            {
                  MasterDialogController.PopController();
            }

            /// <summary>
            /// Syncs the refresh action handler.
            /// </summary>
            /// <param name="sender">The sender.</param>
            /// <param name="args">The <see
cref="com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroupEventArgs"/>
instance containing the event data.</param>
            private void RefreshActionHandler(IActionGroup sender,
IActionGroupEventArgs args)
            {
                  //var ctrl =
MasterDialogController.CreateController(typeof(ReportPreviewController),
ReturnParameters);
                  //MasterDialogController.PushController(ctrl);
                  // Do something to refresh the UI with actual data.
            }

            /// <summary>
            /// Syncs the send logs action handler.
            /// </summary>
            /// <param name="sender">The sender.</param>
            /// <param name="args">The <see
cref="com.sapcd.mobileclient.dialog.Interfaces.Common.IActionGroupEventArgs"/>
instance containing the event data.</param>
```

```
            private void SendLogsActionHandler(IActionGroup sender,
IActionGroupEventArgs args)
            {
                  //var ctrl =
MasterDialogController.CreateController(typeof(ReportPreviewController),
ReturnParameters);
                  //MasterDialogController.PushController(ctrl);
                  // Do something to log the current data to a file or somthing
else.
            }

            #endregion ActionHandlers

      }
}
```

IV. AboutView Extension (Example)

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using com.sapcd.mobileclient.dialog.Interfaces.Common;
using com.sapcd.mobileclient.dialog.Interfaces.Model;
using com.sapcd.mobileclient.dialog.WMForms.UserControls;
using com.sapcd.mobileclient.framework.Localization.Global;
using com.sapcd.mobileclient.framework.Logging;
using SAPCD.MobileSalesClient.Application.UserViewPlugin.About;

namespace MobileSalesExtensibilityTest
{
	public partial class AboutViewExtended : WMFormsDialogUserControl
	{
		#region Member Fields

		private BindingSource bindingSource;
		private AboutViewDataView dataView;
		private readonly MenuItem mnuClear;

		#endregion // Member Fields

		#region Construction / Finalization

		/// <summary>
		/// Initializes a new instance of the <see cref="AboutView"/> class.
		/// </summary>
		public AboutViewExtended()
		{
			InitializeComponent();

			this.mnuClear = new MenuItem();
		}

		#endregion // Construction / Finalization

		#region Overrides

		/// <summary>
		/// Localizes this instance.
```

```
/// </summary>
public override void Localize()
{
        // window title
        Title = "Test";
        // labels
        this.lblCurrentLoggedinUser.Text = "Something";
        this.lblDeviceId.Text = "Something";

        this.lblBatteryStatus.Text = "Something";
        this.lblOsVersion.Text = "Something";
        this.lblIpAdress.Text = "Something";
        this.lblMemoryTotal.Text = "Something";
        this.lblMemoryFree.Text = "Something";
        this.lblDSDVersion.Text = "SAP DSD";

        this.mnuClear.Text = "Clear";

        // buttons
        this.btnBack.Text = "Back";
        // others
        this.headerBarAboutView.LblTitle.Text = "About";

        base.Localize();
}

/// <summary>
/// Registers the data view.
/// </summary>
/// <param name="dv">The dv.</param>
public override void RegisterDataView(IDataView dv)
{
        this.dataView = (AboutViewDataView)dv;
        base.RegisterDataView(dv);
}

/// <summary>
/// Registers the data binding.
/// </summary>
public override void RegisterDataBinding()
{
```

```
                try
                {
                        bindingSource = new BindingSource
                        {
                                DataSource = this.dataView
                        };
                        lblDeviceIdText.DataBindings.Add(new Binding("Text",
bindingSource, "LblDeviceIdText", true, DataSourceUpdateMode.OnPropertyChanged));
                        lblCurrentLoggedinUserText.DataBindings.Add(new
Binding("Text", bindingSource, "LblCurrentLoginUserText", true,
DataSourceUpdateMode.OnPropertyChanged));
                        lblBatteryStatusText.DataBindings.Add(new Binding("Text",
bindingSource, "LblBatteryStatusText", true, DataSourceUpdateMode.OnPropertyChanged));
                        lblOsVersionText.DataBindings.Add(new Binding("Text",
bindingSource, "LblOsVersionText", true, DataSourceUpdateMode.OnPropertyChanged));
                        lblIpAdressText.DataBindings.Add(new Binding("Text",
bindingSource, "LblIpAdressText", true, DataSourceUpdateMode.OnPropertyChanged));
                        lblMemoryTotalText.DataBindings.Add(new Binding("Text",
bindingSource, "LblMemoryTotalText", true, DataSourceUpdateMode.OnPropertyChanged));
                        lblMemoryFreeText.DataBindings.Add(new Binding("Text",
bindingSource, "LblMemoryFreeText", true, DataSourceUpdateMode.OnPropertyChanged));
                        lblDSDVersionText.DataBindings.Add(new Binding("Text",
bindingSource, "LblDSDVersionText", true, DataSourceUpdateMode.OnPropertyChanged));
                }
                catch (Exception e)
                {
                        //Log exception
                        Logger.Singleton.Log(e);
                }

                base.RegisterDataBinding();
        }

        /// <summary>
        /// De-registers the data binding.
        /// </summary>
        public override void DeregisterDataBinding()
        {
                lblDeviceIdText.DataBindings.Clear();
                lblCurrentLoggedinUserText.DataBindings.Clear();
                lblBatteryStatusText.DataBindings.Clear();
                lblOsVersionText.DataBindings.Clear();
```

```
                lblIpAdressText.DataBindings.Clear();
                lblMemoryTotalText.DataBindings.Clear();
                lblMemoryFreeText.DataBindings.Clear();
                lblDSDVersionText.DataBindings.Clear();

                bindingSource = null;

                base.DeregisterDataBinding();
            }

            /// <summary>
            /// Registers the action.
            /// </summary>
            /// <param name="key">The key.</param>
            /// <param name="ag">The ag.</param>
            public override void RegisterAction(Enum key, IActionGroup ag)
            {
                AboutViewController.ActionKeys tmpKey =
(AboutViewController.ActionKeys)key;
                switch (tmpKey)
                {
                    case AboutViewController.ActionKeys.Back:
                    ag.Add(this.btnBack);
                    break;

                    case AboutViewController.ActionKeys.Refresh:
                    ag.Add(this.mnuClear);
                    break;

                    case AboutViewController.ActionKeys.SendLogs:
                    break;
                }

                base.RegisterAction(key, ag);
            }

            /// <summary>
            /// Gets the menu items for the current dialog / view.
            /// Only two items Send and Clear
            /// </summary>
            protected override List<MenuItem> DialogMenu
```

```
            {
                  get
                  {
                        if (this.menuItems.Count == 0)
                        {
                              this.menuItems.Add(mnuClear);
                        }
                        return this.menuItems;
                  }
            }

            #endregion // Overrides


      }
}
```

V. Default DALs / Managers

```
// Document Number Manager Registration

Register<IDocumentNumberManager>(typeof(DocumentNumberManager));
                    Register<IDocumentNumberDAL>(typeof(DocumentNumberDAL));

                    //Collection DAL Registration
                    Register<ICollectionDAL>(typeof(CollectionDAL));

                    //Collection Manager Registration
                    Register<ICollectionManager>(typeof(CollectionManager));

                    //Material Manager Registration
                    Register<ICustomerDAL>(typeof(CustomerDAL));

                    // Customer Manager Registration
                    Register<ICustomerManager>(typeof(CustomerManager));

                    // Pricing Manager Registration
                    Register<IPricingManager>(typeof(PricingManager));

                    // Customizing DAL Registration
                    Register<ICustomizingDAL>(typeof(CustomizingDAL));
                //Register<ICustomizingManager>(typeof(CustomizingManager));

                    // Material Manager Registration
                    Register<IMaterialManager>(typeof(MaterialManager));

                    // Material DAL Registration
                    Register<IMaterialDAL>(typeof(MaterialDAL));

                    // Delivery Item Manager Registration
                    Register<IDeliveryDAL>(typeof(DeliveryDAL));
                    Register<IDeliveryManager>(typeof(DeliveryManager));

                    // Invoice Component registration
                    Register<IInvoiceManager>(typeof(InvoiceManager));

                    //Tour Status Manager Registration
                    Register<ITourManager>(typeof(TourManager));
```

```
                    //Tour DAL Registration
                    Register<ITourDAL>(typeof(TourDAL));

                    //Check Out Check In DAL Registration
                    Register<ICOCIDAL>(typeof(COCIDAL));

                    //Check Out Check In Manager Registration
                    Register<ICOCIManager>(typeof(COCIManager));

                    // Vehicle Manager Registration
                    Register<IQuestionListDAL>(typeof(QuestionListDAL));

                    // Order Manager Registration
                    Register<IOrderManager>(typeof(OrderManager));

                    // ListingExclusion DAL Registration
Register<IListingExclusionDAL>(typeof(ListingExclusionDAL));

                    // ListingExclusion Manager Registration

Register<IListingExclusionManager>(typeof(ListingExclusionManager));

                    // CRMActivity DAL and Manager Registration
                    Register<ICRMActivityDAL>(typeof(CRMActivityDAL));
                    Register<ICRMActivityManager>(typeof(CRMActivityManager));

                    // CRMActivity Journal DAL

Register<ICRMActivityJournalDAL>(typeof(CRMActivityJournalDAL));

                    // Signature Capture Component Registration
                    Register<ISignatureManager>(typeof(SignatureManager));
                    Register<ISignatureDAL>(typeof(SignatureDAL));

                    //Inventory Manager and DAL Registration
                    Register<IInventoryManager>(typeof(InventoryManager));
                    Register<IInventoryDAL>(typeof(InventoryDAL));

                    //Invoice DAL and Manager Registration
                    Register<IInvoiceDAL>(typeof(InvoiceDAL));
```

```
                    Register<IInvoiceManager>(typeof(InvoiceManager));

                    //Physical Unit DAL Registration for Pricing Manager
                    Register<IPhysicalUnitDAL>(typeof(PhysicalUnitDAL));

                    // OCS specific implementation

Register<IOnlineProcessManager>(typeof(OnlineProcessManager));

                    // Field extensions
                    Register<IFieldExtensionDAL>(typeof(FieldExtensionDAL));

                    // Deal conditions
                    Register<IDealConditionsDAL>(typeof(DealConditionsDAL));

Register<IDealConditionManager>(typeof(DealConditionManager));

                    // Manual Discounts
                    Register<IConditionLimitDAL>(typeof(ConditionLimitDAL));

Register<IManualDiscountsManager>(typeof(ManualDiscountsManager));

                    // Messages
                    Register<IMessageDAL>(typeof(MessageDAL));
                    Register<IMessageManager>(typeof(MessageManager));

                    // Password and roles
                    Register<IPasswordDAL>(typeof(PasswordDAL));

        // Order
        Register<IOrderDAL>(typeof(OrderDAL));
        Register<IOrderManager>(typeof(OrderManager));

                    // Expense
                    Register<IExpenseDAL>(typeof(ExpenseDAL));

        //Propose Order
        Register<IProposeOrderDAL>(typeof(ProposeOrderDAL));
        Register<IItemProposalManager>(typeof(ItemProposalManager));

        //Capture-On-Hand
```

```
        Register<ICaptureOnHandDAL>(typeof(CaptureOnHandDAL));

    Register<ICaptureOnHandManager>(typeof(CaptureOnHandManager));

                        // Reason Types
                        Register<IReasonTypeDAL>(typeof(ReasonTypeDAL));

                        //PE Condition Types
                        Register<IPeConditionTypeDAL>(typeof(PeConditionTypeDAL));

                        // App log
                        Register<IApplicationLogDAL>(typeof(ApplicationLogDAL));

                        // Empties Balance
                        Register<IEmptiesBalanceDAL>(typeof(EmptiesBalanceDAL));

                        // Product Hierarchy

    Register<IProductHierarchyDAL>(typeof(ProductHierarchyDAL));

                        // Print Report
                        Register<IPrintReportManager>(typeof(PrintReportManager));

                        // Campaign Determination
                        Register<ICampaignDAL>(typeof(CampaignDAL));

                        //Regions
                        Register<IRegionDAL>(typeof(RegionDAL));
```

VI. Data View DAL Usage (Example)

```
/// <summary>
/// Selects the data.
/// </summary>
/// <param name="selectionData">The selection data.</param>
public override void SelectData(VolatileDataContainer selectionData)
{
	base.SelectData(selectionData);

	if (selectionData != null)
	{
		//mandatory fields
		Tour = selectionData.GetObject(Consts.KEY_CURRENT_TOUR) as
TourBO;
		Visit = selectionData.GetObject(Consts.KEY_CURRENT_VISIT)
as VisitBO;
		Activity =
selectionData.GetObject(Consts.KEY_CURRENT_ACTIVITY) as ActivityBO;

		if (Visit != null)
		{
			CustomerName = Visit.CustomerName;

			// get values if not set in container
			using (MseNamingContext context = new
MseNamingContext())
			{
				ICollectionManager collectionManager =
context.GetInstance<ICollectionManager>();
				ICustomerDAL customerDal =
context.GetInstance<ICustomerDAL>();
				//get customer sales area if not already in
VolatileDataContainer
				if
(selectionData.ContainsKey(Consts.KEY_CURRENT_CUSTOMER_SALES_AREA) &&

selectionData.GetObject(Consts.KEY_CURRENT_CUSTOMER_SALES_AREA) != null)
				{
					CustomerSalesArea =
selectionData.GetObject(Consts.KEY_CURRENT_CUSTOMER_SALES_AREA) as
CustomerSalesAreaBO;
				}
				else
```

```
                                        {
                                                CustomerSalesArea =
customerDal.FindCustomerSalesAreaData(Visit.CustomerNumber, Visit.SalesOrg,

      Visit.DistributionChannel, Visit.Division);
                                        }

                                        //get customer bo if not already in Container
                                        if
(selectionData.ContainsKey(Consts.KEY_CURRENT_CUSTOMER) &&
selectionData.GetObject(Consts.KEY_CURRENT_CUSTOMER) != null)
                                        {
                                                CurrentCustomer =
selectionData.GetObject<CustomerBO>(Consts.KEY_CURRENT_CUSTOMER);
                                        }
                                        else
                                        {
                                                CurrentCustomer =
customerDal.FindCustomer(Visit.CustomerNumber);
                                        }

                                        // get payment information from below variable
could be normal sales area or sold to sales area
                                        CustomerSalesAreaBO soldToSalesArea =
CustomerSalesArea;

                                        //If partner functions are enabled, use sold
to number for getting payment information
                                        if
(CustomizingUtil.CustomizingCountry().AdditionalPartnerDownload)
                                        {
                                                CustomerPartnerFuncBO partnerFunctions
= context.GetInstance<ICustomerDAL>()

       .FindCustomerPartnerFunc(Visit.CustomerNumber, Visit.SalesOrg,

                                                 Visit.DistributionChannel,
Visit.Division);
                                                if (partnerFunctions != null)
                                                {
```

```
                                                //if partner function is found,
load Sales Area of Sold-To customer
                                                SoldToCustomerSalesArea =
customerDal.FindCustomerSalesAreaData(partnerFunctions.SoldToNumber, Visit.SalesOrg,


      Visit.DistributionChannel, Visit.Division);
                                                if (SoldToCustomerSalesArea !=
null)
                                                {
                                                      soldToSalesArea =
SoldToCustomerSalesArea;
                                                }
                                          }
                                    }

                                    //get enabled payment method keys
                                    if (EnabledPaymentMethodKeys == null ||
EnabledPaymentMethodKeys.Count == 0)
                                    {
                                          EnabledPaymentMethodKeys =
collectionManager.GetAllowedPaymentMethodsOfCustomerSalesArea(soldToSalesArea);
                                    }

                                    //get PaymentMethod payment method from
customer sa. if invalid, set first enabled payment method.
                                    if (CustomerSalesArea != null)
                                    {
                                          PreferredPaymentMethodKey =
soldToSalesArea.PaymentMethod;
                                    }

                                    //get available payment methods
                                    if (AvailablePaymentMethods == null ||
AvailablePaymentMethods.Count == 0)
                                    {
                                          AvailablePaymentMethods =
CustomizingUtil.FindCustomizings(Consts.PYMT_TYPE) as List<CustomizingBO>;
                                    }

                                    //Set payment method display name
                                    if (AvailablePaymentMethods != null)
                                    {
```

```
                                        CustomizingBO prefPaymentMethodCust =
                                                AvailablePaymentMethods.Find(x =>
x.KeyValue.Equals(PreferredPaymentMethodKey));
                                        PaymentMethod = prefPaymentMethodCust
== null ? String.Empty : prefPaymentMethodCust.Description;
                                    }

                                    //if PreferredPaymentMethodKey is not in
EnabledPaymentMEthodKeys (and not null or empty), add it
                                    if
(!string.IsNullOrEmpty(PreferredPaymentMethodKey) &&

!EnabledPaymentMethodKeys.Contains(PreferredPaymentMethodKey))
                                    {

      EnabledPaymentMethodKeys.Add(PreferredPaymentMethodKey);
                                    }

                                    //we need to check if the enabled keys differ
from the available keys in the customizing
                                    IList<string> paymentMethodKeysToRemove = new
List<string>();
                                    foreach (string enabledPaymentMethodKey in
EnabledPaymentMethodKeys)
                                    {
                                        if (AvailablePaymentMethods != null)
                                        {
                                            CustomizingBO
availablePaymentMethodMethod =

      AvailablePaymentMethods.Find(x => x.KeyValue.Equals(enabledPaymentMethodKey));
                                            //if enabled key could not be
found in customizing, remove it from the list and log it
                                            if (availablePaymentMethodMethod
== null)
                                            {

      paymentMethodKeysToRemove.Add(enabledPaymentMethodKey);

      Logger.Singleton.Log(LogLevel.Warn,

                        "Payment method for customer '" + Visit.CustomerNumber + "'
with payment key '" +
```

```
                        enabledPaymentMethodKey + "' could not be found. Please
check this!", "CollectionOverview");
                                                }
                                        }
                                }
                                //Remove keys that do not have a payment
method
                                foreach (string keyToRemove in
paymentMethodKeysToRemove)
                                {

        EnabledPaymentMethodKeys.Remove(keyToRemove);
                                }
                        }
                }
                }

                //if calling controller has passed open items & remaining amount
for these, we do not need to reload these
                if (selectionData != null &&
selectionData.ContainsKey(Consts.KEY_OPEN_ITEM_CHECK_RESULT))
                {
                        openItemCheckResult =
selectionData.GetObject<OpenItemCheckResult>(Consts.KEY_OPEN_ITEM_CHECK_RESULT);
                        openItems = OpenItemCheckResult.OpenItems;
                        remainingAmountsForOpenItems =
OpenItemCheckResult.RemainingAmountsForOpenItems;
                        RefreshOpenItemList(true, false);
                }
                else
                {
                        //nothing given, so load from db
                        RefreshOpenItemList(true, true);
                }
        }
```

www.sap.com/contactsap