



coresuite framework

Developer Guidelines

Marco Schweighauser

E-mail

marco.schweighauser@coresystems.ch

Telephone

+41 (0) 56 444 20 40

Infoline

+41 (0) 848 088 088








Document History

Version	Section	Date	Processor	Description
001	0	16.07.06	A. Hauri	Creation
007	0	11.12.07	M. Schweighauser	Revision of all chapters
008	0	22.05.07	M. Schweighauser	Changed interface and project template chapter

Internal file:

Dokument3

Clues

Icon	Description
	<i>Warning</i>
	<i>Notice</i>
	<i>Example</i>
	<i>Syntax</i>
	<i>Open tasks, not yet defined</i>
	<i>Extension, not yet realised but in development</i>
	<i>Already existing objects and functions of SAP Business One</i>

This document is only for partners, it is available on www.coresuite.ch.

Table of contents

1	Introduction.....	4
1.1	What is the coresuite framework	5
1.2	The Ram problem	5
1.3	What does the coresuite framework provide to developers?	6
1.3.1	What parts of the coresuite framework are realised?	6
1.3.2	Further developments	7
1.3.3	What is a Sip?	7
1.3.4	The coresuite Updater	7
1.3.5	What do I need to develop add-on	8
2	Configure your developing environment.....	9
2.1	Install the libraries	9
2.2	Check the sample projects	9
3	Create your first add-on.....	10
3.1	Create the initial project.....	10
3.2	Create a Sip	11
4	Guidelines.....	12
4.1	Software Design	12
4.2	3-Tier Model.....	12
4.2.1	Data tier	13
4.2.2	Logic tier	13
4.2.3	View tier	13
4.2.4	Coresystems best practise	13
4.3	coresystems DTO/DAO classes	14
4.3.1	Data tier	14
4.3.2	Logic tier	17
4.4	The add-on technology	19
4.4.1	Conventions	19
4.4.2	Add-on interface	20
4.5	Creating forms.....	22
4.5.1	The Xml technology	22
4.5.2	Using Items.....	23
4.5.3	The Event interface.....	23

1 Introduction

This document describes how you can create an add-on that uses the **coresuite framework**. This will be done in a few steps. First it will be shown what the **coresuite framework** actually is and how it works. Second guidelines for programming with the **coresuite framework** are provided, to make sure that your add-on uses the full potential of the **coresuite framework** add-on technology!

The purpose of this document is to support the course for **coresuite framework** developing that **coresystems ag** offers. The document is not thought as a reference for programming with **coresuite framework**. It only shows the key features.

1.1 What is the coresuite framework

coresystems ag evaluates and programs add-ons for SAP AG. For the optimal functionality of these add-ons, a special framework has been developed. **coresuite framework** makes it possible to integrate several extensions (called modules), without activating numerous add-ons at the startup of SAP Business One. The result of this technology is a big performance increase and is making the work with SAP Business One quicker and more comfortable.

With the **coresuite framework** the need of memory can be heavily reduced. Compared to the usual SAP add-on technology, 70MB of memory can be saved per module.

With the live update function, modules can always be updated and new ones can simply be added – without manual installation.

For an overview of the **coresuite framework** technology see figure 1:



Figure 1: **coresuite framework**

1.2 The Ram problem

As it is commonly known the connection to SAP Business One is using up to 70 MB for each add-on. Since SAP Business One normally needs customizations for working with it, the customer must use add-ons. Normally not only one but up to five add-ons! This means each instance of SAP Business One will instantiate $5 \times 70 \text{ MB} = 350 \text{ MB}$ of memory at least. This is easy done on a single user machine, but what if SAP Business One is running on a terminal server. You won't be able to run more then 10 users per machine without a significant amount of built in memory.

This is where **coresuite framework** comes in. Since it only uses one connection per instance, it uses only 1 x 70 MB of memory and provides at the same time the possibility to load several add-ons.

1.3 What does the coresuite framework provide to developers?

The **coresuite Framework** provides a well structured and object orientated API to interact with SAP Business One. It doesn't enhance the functionality of the SDK but makes it easier to use. It provides standardized solutions for SAP Business One SDK workarounds, fastest ways of interaction with Business One and a collection of default classes that are usually needed to create by the developer. Examples:

- Event system for all UI components
- Data Access Objects
- Data Transfer Objects
- Automatic creation of tables when they are needed

and lots of other stuff.

1.3.1 What parts of the coresuite framework are realised?

1.3.1.1 Events

The biggest problem in SAP Business One is the event handling. In the SDK there is one general method that is being called on an SAP Business One event. That was a big problem since you had to use *if then else* structures all the time in these methods. Moreover if the method wasn't written properly SAP Business One was made extremely slow and if the method crashed the add-on has been disconnected from SAP Business One.

The **coresuite framework** provides various separate event handlers, the event handlers can be registered per event on each item. Further on the event handling has been made more intuitive to use and even more efficient. No *if then else* structures are needed anymore.

1.3.1.2 UI items

Further all UI items in SAP Business One are wrapped by the **coresuite framework**. They have been made easier to use and they are now well integrated in a class hierarchy that uses the full potential of derivation.

1.3.1.3 Data management

User Tables and UDO's have been a big problem in the SAP Business One SDK. It wasn't very clear how to create, update and make inserts into them. Remember no one is allowed to write to the database by using SQL. This makes it very difficult and complex to handle data in SAP Business One.

A big part of this data handling has already been simplified by the **coresuite framework**.

User Tables

It is now possible to save complete Data Transfer Objects (DTOs) directly to the database without manually creating a table. Even saving and loading can now be done easily by just calling one method. The **coresuite framework** manages updates, inserts and loading of DTOs without to have knowledge of the data access in SAP Business One.

System Tables

coresuite framework has special Data Access Objects (DAOs). The data access classes can be derived and show the programmer exactly what he has to do to implement a correct data access to system tables. Working with system table DTOs is the same as working with user table DTOs. Therefore the way of data access in SAP Business One has been standardized by the **coresuite framework**.

1.3.1.4 Standard Components

The **coresuite framework** moreover contains various solutions to developer problems in SAP Business One. Examples:

- The normal choose form list can be used for user tables. **coresuite framework** includes one that is dynamic and can be used for any table.
- The record set object of SAP Business One is very slow, therefore **coresuite framework** provides a direct connection for select queries to make them faster and more efficient.
- Debug information management for being able to save the debug output to a log file.

1.3.1.5 Upgrades

A special module called **coresuite updater** is able to download Simple Install Packages (SIPs) directly from our server. These SIPs can contain upgrades for any module. This means bug fixes, upgrades and installations are very easy. Just upload a sip to our server and it is distributed to every user.

1.3.2 Further developments

The goal of **coresuite framework** is it to provide solutions to various problems of SAP Business One. We are thinking about including the following:

- Map DTOs directly to form items.
- Make choose form list more user friendly
- Allow the user to customise grids.
- And lots of other stuff

1.3.3 What is a Sip?

A Sip is a Simply Install Package it contains a complete module. This means all its assemblies and other resource data.

A Sip is similar to a zip file. These Sips are unpacked at the very beginning of the **coresuite framework** execution. All files of the **coresuite framework** are exchangeable except for the CoresuiteAddon.exe file (this are 30 lines of code and have never been changed since their creation). Sips don't need ARD Files and are installed automatically into the SAP Business One user's installation folder. This is achieved by the **coresuite updater**.

1.3.4 The coresuite Updater

The **coresuite updater** is a download and upgrade system. It includes a little client that asks the **coresystems** server (called CoreFetch) for updates. These updates depend on the version of SAP Business One and on the company (identified by the installation id). **coresuite updater** downloads the available updates into the database. By using the database **coresuite updater** can distribute the downloaded Sips to

each client in the company, where they are unpacked and installed. **coresuite updater** also takes care that the installation is done correctly.

1.3.5 What do I need to develop own modules

coresystems ag recommends the following:

- Microsoft Visual Studio 2005 for development
- Microsoft .Net Framework 2.0 as runtime environment
- Use VB.Net/C# to code
- SAP Business One 2005 A
- Try to avoid using SDK directly since **coresuite framework** provides lots of secure methods.
- NEVER use the SAP Business One SDK event handler functions directly.
- Avoid making windows forms for your SAP Business One modules.
- Winrar/Winzip for the Sip creation
- **coresuite** License form SAP Business One

2 Configure your developing environment

In order to be able to develop modules the developer needs to do the steps explained in this chapter.

2.1 Install the libraries

First of all you need the DLLs of **coresuite framework** at a centralized point on your system. Therefore you need to create the following file structure:

c:\coresuiteAddOn\DLL

c:\coresuiteAddOn\SAP

Place the **coresuite framework** DLLs into the DLL folder copy the SAP Business One DLLs SAPbobsCOM.dll and SAPbouiCOM.dll into the SAP folder.

2.2 Check the sample projects

Try to open and run the sample projects, delivered together with the **coresuite framework**. As soon as you are able to successfully run them, you are ready to develop your own project.

3 Create your first module

To start your module development carrier you need to do the following steps.

3.1 Create the initial project

The initial project for you module is quite simple to create. Just follow these steps:

1. Create a library project. Name it like your module name (eg. COR_TestModule)
2. Add the following references to the project (the libraries are in the folder "Samples\DLL")
 - a. Loader.dll
 - b. swissFramework.dll
 - c. SAPbobCOM.dll
 - d. SAPbouicom.dll
3. Implement the coresuiteFramework.Loader.Module.IModule interface in you main class.
4. Put your module start code into the IModule.Run() Method. This method will be called on the start of the module life cycle.
5. Add a second project (Windows Application) to your Microsoft Visual Studio Solution.
6. Add the following references to the second project
 - a. Loader.dll
 - b. Your first project
7. Mark the second project as start up project
8. Delete the Form.cs file.
9. Change the main method of the Program.cs file to this:

```
COR_TestAddOn.COR_TestAddOn addon = new COR_TestAddOn.COR_TestAddOn();
addon.PreInstall();
addon.Install();
addon.Run();

Application.Run();
```

10. Compile the solution and the initial setup is finished.
11. You can now use your second project to run your module in Debug mode. Open SAP B1 and run the second project. You module will now connect to SAP B1 via the SDK connection.

To deploy your module to the customers, you have to create a Simply Install Packe (Sip) as explained in the next chapter.

3.2 Create a Sip

To create a Simply Install Package (Sip) you just copy the assembly DLL of your module and all its resources to a Zip file. Rename the ending of the Zip file to *.sip and that's it.

File structure of a sip (module name: COR_TestModule):

/COR_TestModule.dll

/COR_TestModule/<all resources in this folder>

The Sip file can now be installed with the help of the **coresuite management administration** tool.

4 Guidelines

In the following chapters it is explained what **coresystems ag** thinks is a good development strategy and a good software architecture. Please try to stick to those guidelines for your development. It has been proved that this way is efficient and easy to maintain.

4.1 Software Design

The usual module for SAP Business One uses data from the database and an UI view. This is the reason why **coresystems** uses the N-tier model for developing projects. In a normal case there are 3 tiers that interact with each other.

4.2 3-Tier Model

To explain the 3-tier model the figure 2 shows the sem:

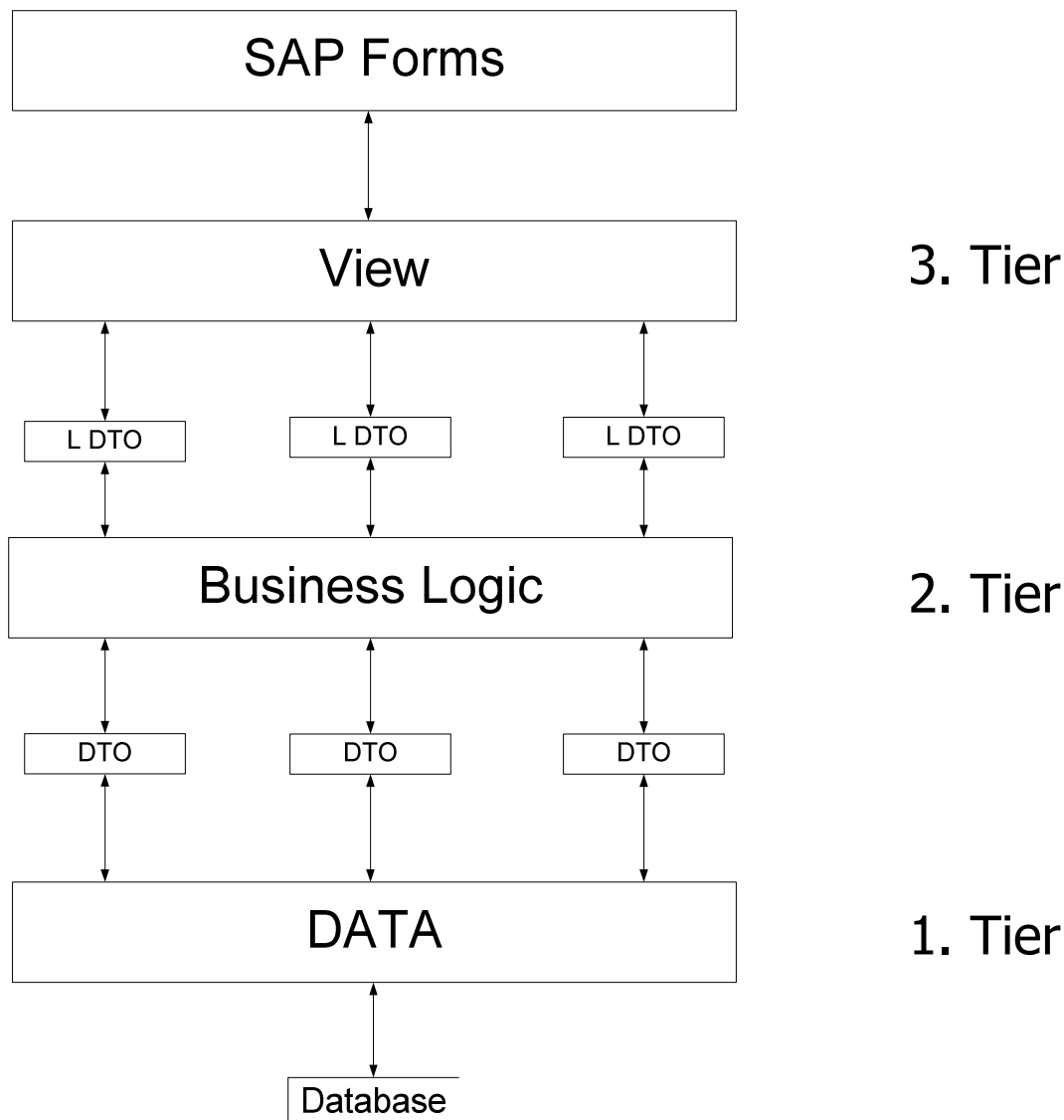


Figure 2: 3-tier model

The tiers shown in the figure 2 are described in the following chapters. The developer makes sure that the data tier only communicates with the logic tier and the view tier only communicates with the logic tier. This makes the interfaces and the communication of the module more transparent to the developer.

4.2.1 Data tier

The data tier is responsible to persist the data into the database and for loading it back into the module. It moreover sends the loaded data to the interface of the logic tier.

4.2.2 Logic tier

The logic tier includes all business processes. All use cases are handled in this tier as well as the behaviour of the non visible items. The DTOs that come from the data tier have been view optimised and the DTOs that are passed from the view tier have been optimised for saving. Every calculation on the DTOs is done in the logic tier.

4.2.3 View tier

The view is the presenting tier that shows the data to the user. This tier can be an SAP Business One form, a windows form or even a webpage.

The view tier just reads the DTOs and shows their values on the screen, reads them back into the DTOs and send them to the logic tier. All behaviour of forms and view data presenting issues is done in the view tier.

4.2.4 Coresystems best practise

coresystems ag uses two types of DTOs for the communication between the tiers.

4.2.4.1 Data tier DTO

The data tier DTOs are used to communicate from the data tier up to the business logic tier and back.

4.2.4.2 Business logic tier DTO

The logic tier DTOs are used to communicate from the business logic tier to the view tier. It is possible to easily convert the data tier DTOs into the logic tier DTOs. A mapper has been written that handles the default mappings for you.

4.2.4.3 Why can't I use data tier DTOs all over?

This might be possible but **coresystems** decided to use data and logic DTOs. We created these two types for the following purpose:

- The data tier DTOs are optimized for saving the data.
- The logic tier DTOs are optimized for displaying them in the view tier.

Since the conversion from a data DTO into a logic DTO and back forth would have meant writing a lot of mapping code we created an automatic mapping. This means except for creating some getters and setter, you as a developer have nothing to do with the mapping if you don't want. But it is possible to interface at any point if you prefer. Moreover we already implemented a data browser class, this data browser class interacts with the data browser buttons of SAP Business One. This means that logic DTOs will be informed when they have to load the next record from the database.

Further more we are developing a mapping functionality that is able to map logic DTOs directly to forms.

4.3 coresystems DTO/DAO classes

Figure 3 shows how **coresystems** implements the DAO DTO theory.

These classes are located in the Namespace:

coresuiteFramework.DI

coresuiteFramework.DI.UserTables

coresuiteFramework.DI.SystemTables

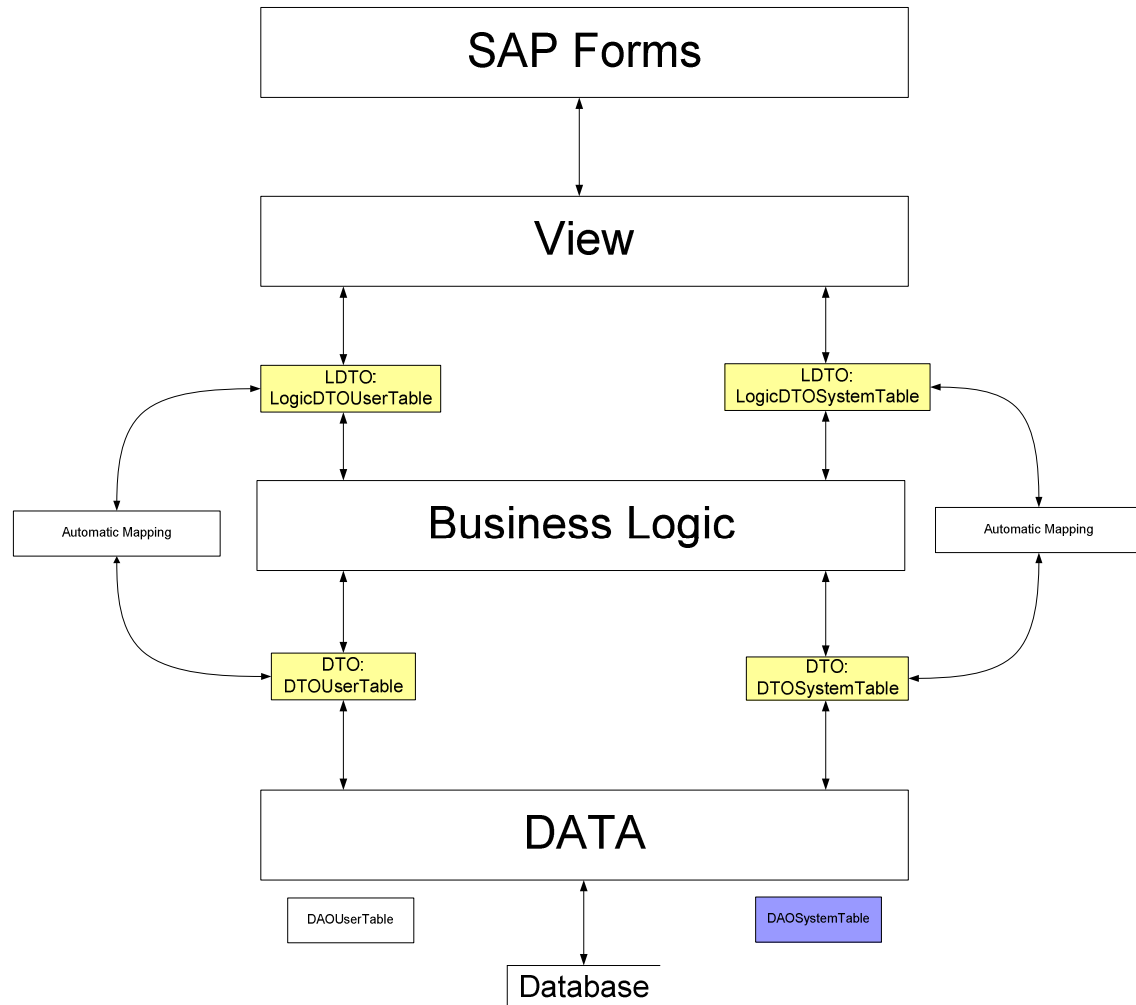


Figure 3: **coresystems** implementation of the 3-Tier model

4.3.1 Data tier

As you see in figure 3 the data tier consists mainly of two classes.

DAOUserTable

This class manages the persistence of DTOs for user tables. This class is fully implemented and doesn't need any programming to work.

DAOSystemTable

This class is an abstract class where you need to implement a lot, this means if you want to connect to a SAP Business One table you need to use the DI interface of the SDK. You have to write the save and load

The example below shows you how you should implement a DAO class to use the system table AdditionalExpenses.

```
using System;
using System.Collections.Generic;
using System.Text;
using coresuiteFramework.DI;

namespace SWA_CT.Data.DAO
{
    class AdditionalExpenses : coresuiteFramework.DI.SystemTables.DAOSystemTable
    {
        //constructors
        public AdditionalExpenses(Data.DTO.AdditionalExpenses dto) : base(dto) { }

        //override methods
        public override void Save()
        {
            throw new coresuiteFramework.DI.DAOException("Save AdditionalExpenses not implemented!");
        }
        public override void Load()
        {
            Data.DTO.AdditionalExpenses dto = (Data.DTO.AdditionalExpenses)this.DTO;
            try
            {
                SAPbobsCOM.AdditionalExpenses addEx = GetAdditionalExpenses();
                dto.DBIdent = addEx.ExpnsCode.ToString();
                dto.Name = addEx.Name;
                this.DTO = dto;
            }
            catch (System.Runtime.InteropServices.COMException ex)
            {
                throw new coresuiteFramework.DI.DAOException("Load AdditionalExpenses not possible: " +
                    ex.Message);
            }
        }
        public override List< coresuiteFramework.DI.IDTO> GetByWhereClause(string whereString)
        {
            List< coresuiteFramework.DI.IDTO> dtoList = new List<IDTO>();
            SAPbobsCOM.Recordset rs = (SAPbobsCOM.Recordset)
                coresuiteFramework.B1Connector.GetB1Connector().Company.GetBusinessObject(SAPbobsCOM.BoObjectTypes.BoRecordset);
            rs.DoQuery("SELECT ExpnsCode, ExpnsName FROM OEXD");
            while (!rs.EOF)
            {
                Data.DTO.AdditionalExpenses dto = new Data.DTO.AdditionalExpenses();
                dto.DBIdent = rs.Fields.Item("ExpnsCode").Value.ToString();
                dto.Name = rs.Fields.Item("ExpnsName").Value.ToString();
                dtoList.Add(dto);
                rs.MoveNext();
            }
            return dtoList;
        }

        //private methods
        private SAPbobsCOM.AdditionalExpenses GetAdditionalExpenses()
        {
            Data.DTO.AdditionalExpenses dto = (Data.DTO.AdditionalExpenses)this.DTO;
            SAPbobsCOM.Company company = coresuiteFramework.B1Connector.GetB1Connector().Company;
            SAPbobsCOM.AdditionalExpenses addExp =
                (SAPbobsCOM.AdditionalExpenses)company.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oAdditionalExpenses);
            try
            {
                addExp.GetKey(int.Parse(dto.DBIdent));
                return addExp;
            }
            catch (System.Runtime.InteropServices.COMException ex)
            {
                throw new coresuiteFramework.DI.DAOException("AdditionalExpense not found: " +
                    ex.Message);
            }
        }
    }
}
```

4.3.1.1 DTOUserTable

The DTOUserTable class is an abstract class. All your DTO's that access user tables have to derive from this class. You have to add properties that you want to map to the database. Each property, which has to be mapped to the database, has to have the attribute [ToPersist]. The class its self should have a [Tablename("<name>")] attribute that describes the name of the table you want to access.

The example below should show how it works:

```
using System;
using System.Collections.Generic;
using System.Text;
using coresuiteFramework.DI;

namespace SWA_CT.Data.DTO
{
    [Tablename("SWA_CT_RESS_STATUS")]
    [Description("coreTime Ressource Status")]
    class ResourceStatus : coresuiteFramework.DI.UserTables.DTOUserTable
    {
        private string description;
        [ToPersist]
        [Description("Description")]
        [FieldType(FieldTypes.Alpha)]
        [Size(30)]
        private string Descript
        {
            get { return description; }
            set { description = value; }
        }
        public string Description
        {
            get { return description; }
            set { description = value; }
        }
    }
}
```



Hint: be aware that this class will contain a code property that returns the unique identification of the current record set.



Warning: Make sure that you name your class with your namespace prefix to avoid conflicts eg.: cor_swa_tabledesc

4.3.1.2 DTOSystemTable

The DTOSystemTable class is an abstract class. This class contains all properties that you want to map to the database. Since you are handling the persistency by your self no attribute for the properties are necessary. The Example below shows how such a class could look like.

```
using System.Text;
using coresuiteFramework.DI;

namespace SWA_CT.Data.DTO
{
    [Tablename("OCRD")]
    class BusinessPartner : coresuiteFramework.DI.SystemTables.DTOSystemTable
    {
        //fields
        private string cardCode;
        public override string DBIdent
        {
            get { return cardCode; }
            set { cardCode = value; }
        }

        private string cardName;
        public string CardName
        {
            get { return cardName; }
            set { cardName = value; }
        }
    }
}
```



```

private string address;
public string Address
{
    get { return address; }
    set { address = value; }
}

private string zipCode;
public string ZipCode
{
    get { return zipCode; }
    set { zipCode = value; }
}

private string city;
public string City
{
    get { return city; }
    set { city = value; }
}

private string phone1;
public string Phone1
{
    get { return phone1; }
    set { phone1 = value; }
}

public Dictionary<string, Data.DTO.ContactEmployee> ContactEmployees
{
    get {
        return ((Data.DAO.BusinessPartner)DAO).GetContactEmployees();
    }
}

//override methods
public override void Save()
{
    DAO.Save();
}

public override void Load()
{
    DAO.Load();
}

public override coresuiteFramework.DI.IDAO DAO
{
    get { return new Data.DAO.BusinessPartner(this); }
}
}
}

```



Hint: Be aware that you need to make sure that the DBindent is set correctly for uniquely identifying the record.

4.3.2 Logic tier

This tier contains two classes that are primary needed for the data mapping. If you need a special mapping for your data DTOs to your logic DTOs you need to implement the mapping yourself.

4.3.2.1 LogicDTOUserTable<DataDTO>

The LogicDTOUserTable<DataDTO> class needs to be implemented, to automatically map the DTOUserTable object to the LogicDTOUserTable object. This means the properties of these two classes are mapped to each other without any coding of the developer. You need to add the [map] attribute to each property that you want to map from your data DTO to your logic DTO. Since this is an abstract generic class you need to declare the type of your DTO when you derive from it.

Moreover it is important that you provide 3 constructors. Theses constructors need to call the base class constructor. As you see in the example. If you don't do that you might experience a wrong behaviour when

you start linking DTO objects.

Example:

```
using System;
using System.Collections.Generic;
using System.Text;
using coresuiteFramework.DI;

namespace SWA_CT.Logic.DTO
{
    class RessourceStatus :
coresuiteFramework.DI.UserTables.LogicDTOUserTable<Data.DTO.RessourceStatus>
    {
        //constructors
        public RessourceStatus() { }
        public RessourceStatus(string id): base(id) {}
        public RessourceStatus(Data.DTO.RessourceStatus dto) : base(dto) { }

        //properties
        private string description;
        [Map]
        public string Description
        {
            get { return description; }
            set { description = value; }
        }
    }
}
```

4.3.2.2 LogicDTOSystemTable<DataDTO>

The LogicDTOSystemTable<DataDTO> is implemented the same way as the LogicDTOUserTable is implemented. This means it will map the properties of those classes. Make sure that u add a [map] attribute to each property you want to be mapped.

Moreover it is important that you provide 3 constructors. Theses constructors need to call the base class constructor. As you see in the example. If you don't do that you might experience a wrong behaviour when you start linking DTO objects.

Example:

```
using System;
using System.Collections.Generic;
using System.Text;
using coresuiteFramework.DI;

namespace SWA_CT.Logic.DTO
{
    class BusinessPartner :
coresuiteFramework.DI.SystemTables.LogicDTOSystemTable<Data.DTO.BusinessPartner>
    {
        //constructors
        public BusinessPartner() { }
        public BusinessPartner(string id): base(id) {}
        public BusinessPartner(Data.DTO.BusinessPartner dto) : base(dto) { }

        //properties
        public string CardCode
        {
            get { return this.DbIdentification; }
            set { DbIdentification = value; }
        }

        private string cardName;
        [Map]
        public string CardName
        {
            get { return cardName; }
            set { cardName = value; }
        }

        private string address;
        [Map]
        public string Address
        {

```

```

        get { return address; }
        set { address = value; }
    }

    private string zipCode;
    [Map]
    public string ZipCode
    {
        get { return zipCode; }
        set { zipCode = value; }
    }

    private string city;
    [Map]
    public string City
    {
        get { return city; }
        set { city = value; }
    }

    private string phone1;
    [Map]
    public string Phone1
    {
        get { return phone1; }
        set { phone1 = value; }
    }

    public Dictionary<string, Logic.DTO.ContactEmployee> ContactEmployees
    {
        get
        {
            Dictionary<string, Data.DTO.ContactEmployee> dtoDataList =
this.DataDTO.ContactEmployees;
            Dictionary<string, Logic.DTO.ContactEmployee> dtoLogicList = new Dictionary<string,
ContactEmployee>();
            foreach (Data.DTO.ContactEmployee ceData in dtoDataList.Values)
            {
                dtoLogicList.Add(ceData.DBIdent, new Logic.DTO.ContactEmployee(ceData));
            }
            return dtoLogicList;
        }
    }
}

```

4.4 The module technology

The module technology is similar like the add-on system you might know from other programs. **coresuite framework** will search trough all DLLs that are in the add-on folder and implement a special interface. If a class implements that interface, the class is loaded into the **coresuite framework**. All these modules do finally interact with SAP Business One over the event methods. These methods have been wrapped to make sure that **coresuite framework** wont crash when one a module crashes.

4.4.1 Conventions

Most important is: **Always stick to your Namespace! Do never forget your prefix for tables, DLLs and folders!**

Further more if you need to use the old-style event interface, connect your methods to the following event handler Class:

coresuiteFramework.Global

In terms of bubble event make sure if you set it to true or false always use a logic &.

Exmple:

```
bubbleevent &= <true|false>;
```

If you don't do that you could interfere with another module which will create almost untraceable errors. That could crash the complete **coresuite** add-on and all its functionalities. Your add-on will therefore not be certified.

4.4.2 Module interface

The interface is specified as follow:

```
namespace coresuiteFramework.Loader.Module
{
    public interface IModule
    {
        string ModuleGuid { get; }
        string ModuleVersion { get; }
        string ModuleName { get; }
        string ModuleInfoLink { get; }

        void CreateMenu(SwissAddonFramework.UI.Components.MenuItem menuItemConfiguration);
        void CompanyChanged();
        void LanguageChanged();
        bool PreInstall();
        void Install();
        void Run();
        void Terminate();
    }
}
```

And here is an example how it is implemented:

```
public class TestModule: coresuiteFramework.Loader.Module.IModule
{
    public string ModuleGuid
    {
        get { return "COR_TestAddOn"; }
    }

    public string ModuleVersion
    {
        get { return "1.0"; }
    }

    public string ModuleName
    {
        get { return "coresuite administration"; }
    }

    public string ModuleInfoLink
    {
        get { return "http://www.coresuite.ch/"; }
    }

    public void CreateMenu(SwissAddonFramework.UI.Components.MenuItem menuItemConfiguration)
    {
        // Create the AddOn menu here
    }

    public void CompanyChanged()
    {
    }

    public void LanguageChanged()
    {
    }

    public bool PreInstall()
    {
        return false;
    }

    public void Install()
    {
    }
}
```

```
public void Run()
{
    // Add your main code here
}

public void Terminate()
{
}
}
```

4.4.2.1 string ModuleGuid

This property returns the guid of your module. The guid must not change and is a unique identifier of your module. The guid has to have your SAP AddOn identifier as prefix (eg. COR for **coresystems**).

4.4.2.2 string ModuleVersion

This property returns the version number of your module. You can change the version number as often as you like. The version number is displayed in the **coresuite management administration**.

4.4.2.3 string ModuleName

The ModuleName property returns the name of your module and it is displayed in the **coresuite management administration** as well as on the start-up. You can choose the name displayed by yourself.

4.4.2.4 string ModuleInfoLink

With the ModuleInfoLink property you get the possibility to define a product information link. The product information link is displayed in the **coresuite management administration** and helps the customer to get more information about a module.

4.4.2.5 void CreateMenu()

All the menu item creation code has to be put into this method. The CreateMenu() method is called during the start of the module and on every language and company change.

4.4.2.6 void CompanyChanged()

This method is called when the company is changed.

4.4.2.7 void LanguageChanged()

This method is called when the SAP Business One language is changed. Your module might support multiple languages. Here is the place to change the language of your module.

4.4.2.8 bool PreInstall()

This method is called at the very beginning of the execution. Only do data definitions tasks in this method (means create tables, user fields and stuff). Make sure that if you require a restart after installing your tables and fields return "true" if you don't require a restart return "false".

Make sure this part is only executed once. Never execute it again if your data is consistent. This code must as well be able to recover an incorrupt installation.

4.4.2.9 void Install()

This method is called on every start before the Run method is started. Put everything in here that is not changing the database definition (inserts updates and stuff is ok).

Make sure that this method is only executed when the module is not installed or the installation is inconsistent. Check your data structures for the needed parameters/definitions/data and try to keep the execution time and the execution of the install commands to a minimum. This saves starting time and installation is normally only needed once. Please try not to ask the user for anything, because the install should be done automatically. If you need to know parameters try to figure them out from the context or give the user a configuration form in the Menu of SAP Business One. Make sure u can recover from a buggy installation after the next restart and catch all exceptions of your module.

4.4.2.10 void Run()

This function is called to start the module. Put everything here that would normally be in a Main method.

4.4.2.11 void Terminate()

This method is called if the **coresuite framework** is closed and executes all the things that needs to be done before shutdown your module.

4.5 Creating forms

This chapter gives an introduction how to create SAP Business One forms and items by using the **coresuite framework**.

4.5.1 The Xml technology

One of the SAP Business One disadvantages is that it is very slow to create forms and items by code. The **coresuite framework** provides the ability to generate the Xml code for you and to handle the batch load. Therefore you can load forms from a file or by code. Mixing of the two different ways is also possible. For example you can load a form with a file and add additional items to it by creating them manually in the code.

4.5.1.1 Usage of a Xml file

Create your forms and items with the help of the SAP B1 Screen Painter or by manually create the Xml file.

The following example shows how you can load a form.

```
Form myForm = Form.GetFormFromFile("myForm.xml");
```

4.5.1.2 Create forms and items by code

You can create your forms and items within your code and the **coresuite framework** will create internally an Xml file and batch load it to SAP Business One. You don't have to worry anymore about writing an Xml file.

The following example shows how to create a form and a button item.

```
Form myForm = Form.CreateNewForm("myFormType", "myFormUniqueId");
myForm.Title = "My first own form";

Button myButton = Form.CreateNew();
myButton.UniqueId = "myButtonUniqueID";
myButton.Caption = "My button";

myForm.AddItem(myButton);
myForm.Load();
```

The items have some default SAP like values and you don't need to assign things like width or height to a button.

The `myForm.Load` expression creates the Xml file and loads the form into SAP Business One forms. Therefore the form and the button exist only after the Load call.

The following example shows how to mix the two different ways of creating a form.

```
Form myForm = Form.GetFormFromFile("myForm.xml");

Button myButton = Form.CreateNew();
myButton.UniqueId = "myButtonUniqueID";
myButton.Caption = "My button";

myForm.AddItem(myButton);
myForm.Load();
```

4.5.2 Using Items

You can use the items like the SAP Business One items. Some properties of an item can be used before the item is loaded in SAP Business One, to set settings like width or caption. Please take a look at the **coresuite framework** SDK Help to get informed about which item property throws an exception if you set a setting before loading the item in SAP Business One.

Most items give you the following possibilities to create or use them:

- `Item.CreateNew` - Creates a totally new item
- `Item.GetFromUID` - Returns an existing item. This is useful if you want to use system items or items which already have been load to SAP Business One.

4.5.3 The Event interface

One of the big advantages of the **coresuite framework** is the ability to register events to a specific item and use the C# delegate functionality. Therefore no more *if - then* or *select - case* is needed for your item handler method. Because of the fact that every item has different events you can only register the useful events and don't have to worry about which item supports which events.

The following example shows how to register an item pressed event to a button.

```
myButton.AddHandler_ItemPressed(FormModes.All, myBeforeEvent, myAfterEvent);
```

```
public void myBeforeEvent(ItemPressed e)
{ /* Do something */ }

public void myAfterEvent(ItemPressed e)
{ /* Do something */ }

// It is also possible to register no bubble events:
myButton.AddHandler_ItemPressed(FormModes.All, myNoBubbleEvent);

public void myNoBubbleEvent (ItemPressed e)
{ /* Do something */ }

// It you want to use only a before or after event, set the
// other event to null:
myButton.AddHandler_ItemPressed(FormModes.All, myOnlyBeforeEvent, null);

public void myOnlyBeforeEvent (ItemPressed e)
{ /* Do something */ }
```

The first parameter always specifies the form mode where the event is raised. This gives you the possibility to have different event handlers for different form modes.

There is one special event if you want an event for all forms of a special type. This is useful if you want to add a button to all forms of a special type. For example you want to add a "Search" button to the employee form.

```
Form.AddHandler_Load("employeeFormType", myAfterEvent);

public void myAfterEvent(ItemPressed e)
{
    Button searchButton = Button.CreateNew();
    searchButton.UniqueID = "mySearchButton";
    searchButton.Caption = "Search";
    e.Form.AddItem(searchButton);
    e.Form.Load();
}
```