This unit is a short outline and will give you an overview on component level.

In addition it will show how SAP uses the SDK for extensions (i.e. „Add-Ons‟) to SAP Business One.
The Data Interface API: Unit Objectives

At the conclusion of this unit, you will be able to:

- Describe what the Data Interface API is
- Explain how the Data Interface API exchanges data with SAP Business One
- Use the most important objects of the Data Interface API
The Data Interface API: Business Example

Due to the specified requirements you need to add functionality outside the SAP Business One application.

For this purpose, you use the SAP Business One Data Interface API.
The Data Interface API: Unit Overview Diagram

**The Data Interface API**

<table>
<thead>
<tr>
<th>Topic 1: DI API Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: Business Objects</td>
</tr>
<tr>
<td>Topic 3: Non-Business Objects</td>
</tr>
<tr>
<td>Topic 4: Meta Data Objects</td>
</tr>
<tr>
<td>Topic 5: DI API Services</td>
</tr>
<tr>
<td>Topic 6: Java Connector (optional)</td>
</tr>
<tr>
<td>Topic 7: DI Server (optional)</td>
</tr>
</tbody>
</table>
DI API Introduction: Topic Objectives

At the conclusion of this topic, you will be able to:

- Explain the architecture of the DI API
- Describe how compatibility is implemented
- Categorize available objects
- Explain the key features of DI API
- Describe details regarding connection to a company
This slide provides details about the software architecture of the data interface API (DI API): The business functions are included in an implementation layer (OBServerDLL.DLL). The DLL is based on existing source code of the SAP Business One client, that is, the business objects of the SAP Business One client were copied to this DLL.

You can access the business objects of the SAP Business One client by addressing the interface layer, which is represented by the SAPbobsCOM.DLL. In addition to the existing business objects, you can also address more generic objects such as the RecordSet object.
The Data Manager stores temporary object data, converts object data to internal data formats, retrieves data from the database, and controls the database transactions.

The Schema Generator creates XML schemas based on object interface descriptions. The schema generator also creates object validation lists.

The DI Core, which is the main component of the DI API, performs all the data logic operations.

The COM Interface provides the interface to the add-on application.

The DI API uses the OBSERVERDLL.dll component that performs all the business logic operations. (The OBSERVERDLL.dll component is not a part of the DI API package, but is distributed with the SAP Business One application.)

The DI API is a wrapper to the OBSERVERDLL.dll

Please note:
- Not only the same business logic as you can find in the SAP Business One application applies when DI API is used, but also all the permissions set for the user will allow or disallow particular transactions – just as it will be in the application!
DI API version should be equal to the company version or smaller than that. (For example: If the company version is 8.8 than the DI API version can be 2007 or 8.8) → Maximum = the company version

Observer DLL version will be equal to company version.

In detail (if the referenced version of DI API is installed on the client PC):

- An Add-On application using DI API 2007 can connect to any company database of version 2007 or 8.8
- An Add-On application using DI API 8.8 can connect to any company database of version 8.8, but not of version 2007

Please note:

For the RecordSet object compatibility may change due to incompatible changes in database structure.
Objects in DI API can be divided into three basic groups:

- Business Objects
- Infrastructure Objects
- Special Objects
A lot of business objects contain collections of additional objects like UserFields and more
Service Type Objects / Services

- DI Services / Service Type objects are meant to reflect the concept of Service-Oriented Architecture (SOA) in the SAP Business One world.
- The DI Services provide interfaces to additional logic within SAP Business One, which is not necessarily encapsulated in a business object.

The main service is CompanyService:

- It allows to manage administrative data of a company.
- For example, you can update the Administration data (OADM) or Company data (CINF) or create new Posting Periods (OACP) or update Finance Periods (OFPR).
DI API Introduction – Infrastructure Objects

The Infrastructure objects do not represent SAP Business One data.

- **Company object**
  - Represents an SAP Business One Company database on Microsoft SQL Server/HANA
  - Use this object to access the other objects in DI API

- **Extended Functionality Objects**
  - **Recordset**
    - Used to run SQL queries and stored procedures
  - **DataBrowser**
    - Enables data navigation trough records of a certain object Type (e.g. business partners) in conjunction with Recordset
  - **SBObob**
    - Exposes extended / supplemental functionality

- **Meta Data Objects**
  - **UserTablesMD** - Create user tables
  - **UserKeysMD** - Define an index for a user table
  - **UserFieldsMD** - Create user fields (into SAP Business One tables or user tables)
  - **UserObjectsMD** - Define User Defined Objects

- The Company object is the main object of the Data Interface API.
- The RecordSet object allows to run SQL queries to retrieve data.
- Re Recordset:
  - Because the database tables are accessed directly, testing (and probably changes) must be done after upgrading SAP Business One because the database structure might have been changed.
  - The DI API Recordset object has nothing to do with e.g. ADO Recordset etc.
The Company object is the main object of the Data Interface API. You have to use a method of the Company object to connect to an existing SAP Business One database. Correspondingly, you can also disconnect your application from that database. When you have established a connection, you can access data in the corresponding SAP Business One database for the Company object.

Using the corresponding methods of the Company object, you can also create logical units of work or global transactions, which span more than one business object.

Moreover, the Company object provides methods to extract a business object.

You can find more information about the components of the Company object in the obsCOM help file.
To run an Add-On application, you must first establish a connection to a database. The code for the connection is fairly simple as shown on this slide.

Follow these steps to establish a connection to a database:

- Define variable for the Company object.
- Initialize the Company object.
- Set connect (server) data.
- Set AddOn identifier
  - you must have a fully-licensed development environment to use this (including SDK Dev license or solution license for your AddOn) - not available in evaluation environment
  - Details will be discussed later
  - Don’t set AddOnIdentifier, if running on evaluation
- Connect to SAP Business One.
- Execute error handling.

To use SAPbobsCOM.DLL, you have to set a reference. In Visual Studio 98, for instance, you can do that in Project → References.

Please note that some properties are optional.
DI API Introduction – Error Handling in DI API

There are two ways you must be prepared to handle errors:

- **Return Code + GetLastError**
  Use the return value of some methods to verify the result of the execution, such as Add, Update, Remove...
  Use GetLastError method of the Company object to retrieve the last error message and code issued by any object related to the Company object

AND

- **Exception Handling**
  Some objects will throw an exception.
  In VB, we can use “On Error GoTo ErrorHandler” to process these errors – or Exception handling (try / catch in .NET incl. VB .NET).

  Exception can be raised by methods and properties (e.g. type mismatch)
DI API Introduction: Topic Summary

You should now be able to:
- Explain the architecture of the DI API
- Describe how compatibility is implemented
- Categorize available objects
- Explain the key features of DI API
- Describe details regarding connection to a company

Connecting has already been practiced in the introduction unit…
DI API Introduction: Exercise

You should create a new Microsoft Visual Studio.NET project for VB.NET and practice the first exercise:

- Connect to a SAP Business One company database using DI API...
The Data Interface API: Unit Overview Diagram

The Data Interface API

Topic 1: DI API Introduction
Topic 2: Business Objects
Topic 3: Non-Business Objects
Topic 4: Meta Data Objects
Topic 5: DI API Services
Topic 6: Java Connector (optional)
Topic 7: DI Server (optional)
Business Objects: Topic Objectives

At the conclusion of this topic, you will be able to:
- Describe what business objects are
- List the most important methods of business objects
- Explain how to read or write a business object from or to an XML file
- Design a transaction involving more than one business object
- Tell how to get notified on changes in business objects
Let us look at the business partner as an example for business objects:

- Besides the object itself and all the properties that represent single data in the record in the database, it contains a larger number of properties that represent "sub-objects" in the database stored in different tables.
- In this case these "sub-objects" represent also the tabs / folders on the Business Partners master data form.
- The layout of other business objects is similar to this.
First of all, we want to add a business partner to the company database (to which we have connected before).

In a first step, you have to create an instance of the business partner object. For this purpose, you use the GetBusinessObject method of the Company object.

Then, you can provide the attributes of the business partner. You have to provide at least the mandatory attributes. In this case you have to provide the CardCode property. The built-in auto-complete procedure completes the default values of the other properties.

In a last step, you call the Add method to create a new business partner record in your Company database.

Please note that GetBusinessObject returns a generic „Object“ that needs to be casted to the real object class in other (non-VB!) programming languages!
Examples of business objects include the following:

- Product tree objects
- Items (represents Master Inventory Items record in SAP Business One)
- Business partners
- Documents (represents the Sales and Purchase documents)
- Payments object

Using the SaveXML method, an object can be extracted and saved as an XML file. XML data can also be imported using the Company object.
Often business objects refer to Line objects.

Examples of Line objects include the following:
- Addresses of business partners (BPAddresses)
- ItemWarehouseInfo contained in Items
- Document lines (Document_Lines object)
- Payment Accounts (Payments.Accounts Object)

Almost all line objects have the following methods:
- Add (add a new line object, for example, add an alternate address for a business partner)
- SetCurrentLine (set the current line within the collection of line objects). The count starts from zero.
Here, we have an example for a line object of the business partner object: You can add several contact employees to the business partner record. To do so, you first have to add a Contact employee row using the corresponding Add method.

In a second step you set the current line in the contact employees array. Then you can provide the contact employee properties.
The Items object represents the Master Inventory Items record in SAP Business One.

The Items object enables you to add, update, or find an items record.
Business Objects: Documents

- The Documents object represents the header of SAP Business One Sales and Purchase Documents
- It contains the master header data for the document such as CardCode, Address, Document Date, Document Total etc.

Documents
  - Lines (Document_Lines)
  - UserFields
  - Browser (DataBrowser)
  - Expenses (DocumentsAdditionalExpenses)
  - WithholdingTaxData
This code sample shows how to add an order containing two lines to the SAP Business One database.
Here you can see how to reference (note rectangles in the code) the order added on slide before in an Invoice to be added to the SAP Business One database right now.

Do you remember how this can be done inside the SAP Business One application?
Other Business Objects: Access Log

Displays access details of users who have logged on and logged off with the SAP Business One client or DI API.

In the log, system administrators and system auditors can check user properties and access states at any time so that they can verify that users in the system are managed correctly and the system offers an adequate level of security.

- Several new properties added to the Users object.
  - UserActionRecord object added
Working with XML – Motivation

A Technique of saving and loading data

XML Advantages

- Enable exchanging large-scale data between SAP Business One company database and customer’s database (regardless of the database type)
- Standard
- Cheap
- Convenient
Taken from the DI API documentation (SDK HelpCenter):

**XmlExportType** – Valid Values:
- **xet_AllNodes**
  Export to XML all fields (both read only and read/write fields) from the database.
  (XML files cannot be read using ReadXml or GetBusinessObjectFromXML.)
- **xet_ValidNodesOnly**
  Export to XML only valid fields that support XML import (read/write fields only) from the database.
  (XML files cannot be read using ReadXml or GetBusinessObjectFromXML.)
- **xet_NodesAsProperties**
  Export to XML all fields as properties from the database.
  (XML files cannot be read using ReadXml or GetBusinessObjectFromXML.)
- **xet_ExportImportMode**
  Export to XML only valid fields that support XML import and export (read/write fields only that do not contain null values) from the database.
  (XML files CAN be read by the ReadXml or GetBusinessObjectFromXML method.)
Working with XML
Delete line/sub-object by XML

- Perform a line/sub-object deletion via an XML interface..
  - Available for:
    - BusinessPartners object
    - Documents object
  - New method `UpdateFromXML` added to the business partner and document objects.

- `UpdateFromXML` Receives and processes the XML content. You can remove sub-object lines from the Document object via the XML file.
You can save business object data in XML format in order to use them outside of SAP Business One. To create an XML file, you call the SaveXML method of the corresponding business object.

```
Dim oBP As SAPbobsCOM.BusinessPartners = _
    oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oBusinessPartners)

oCompany.XmlExportType = SAPbobsCOM.BoXmlExportTypes.xet_ExportImportMode

If (oBP.GetKey("C20000") = False) Then
    MessageBox.Show("Failed to find the business partner")
Else
    oBP.SaveXML("c:\temp\BP_" + oBP.CardCode + ".xml")
End If
```
When reading master data items from an XML file, you can use several methods of the Company object to access the type and the number of items in the XML file:

- GetXMLelementCount returns the number of items in the XML file.
- GetXMLobjectType retrieves the item type of a specific item in the XML file.
- GetBusinessObjectFromXML returns the attributes of a specific business object.
When a data operation is performed on a business object, a transaction is started. The SAP Business One database uses transactions to keep the data consistent. If the operation is successful, then a Commit operation is issued and the data is saved. If the operation fails, then a rollback operation is started and the data is discarded. If the data operation is performed on a single business object, all this is done automatically.

If you want to perform database actions that must be divided into several steps, you can use the StartTransaction method to start a series of operations.

When a global transaction is started with StartTransaction, the business objects use this global transaction. If one of the business objects fails during any process, the transaction ends and an automatic rollback operation is started. When the transaction is successful, you must use the EndTransaction method to free the locked records and allow other users access to them.

Use the „InTransaction“ property in case you are not sure about the status of the transaction.
If you use the StartTransaction method you have to commit or roll back the transaction using the EndTransaction method… if nothing went wrong in between.
How to get notified on changes in business objects…

Motivation

- There are no DI API data-driven notifications (only FormData events in the UI API – see UI API presentation)
- Adding SQL triggers or Stored Procedures at the database level is not permitted!

Solution

- Add some code inside the stored procedures called SBO_SP_TransactionNotification (or SBO_SP_PostTransactionNotice).
- The DI EventService tool (on SDN) proposes a ready to use solution based on the SBO_SP_TransactionNotification.

Important remarks:

- The code within the stored procedure runs in database context – i.e. outside an Add-On or DI API-based application...
- If a transaction includes further transactions in the background (e.g. A/R Invoice creates Journal Entry in the background) only information about the “top-level” transaction may get sent to the stored procedure!

- DIEventService: https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/53cefa6a-0a01-0010-cd8e-e7c189cb6519
- SBO_SP_TransactionNotification article:  
  https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/e991e2b9-0901-0010-0395-ef5268b00aaf
- Links may have changed in the meantime though…
- Any synchronization issues – or issues with credentials will have to be considered carefully; usually registering the incoming „events“ and processing them asynchronously should resolve that issue – just like it is handled in the SAP Business One integration package.
Business Objects: Topic Summary

You should now be able to:
- Describe what business objects are
- List the most important methods of business objects
- Explain how to read or write a business object from or to an XML file
- Design a transaction involving more than one business object
- Tell how to get notified on changes in business objects
Business Objects: Exercise

You now should:
- Work with Business Objects in general
- Use the XML capabilities
- Practice Transaction handling along the exercises at the end of this unit...
## The Data Interface API: Unit Overview Diagram

<table>
<thead>
<tr>
<th>The Data Interface API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1: DI API Introduction</td>
</tr>
<tr>
<td>Topic 2: Business Objects</td>
</tr>
<tr>
<td><strong>Topic 3: Non-Business Objects</strong></td>
</tr>
<tr>
<td>Topic 4: Meta Data Objects</td>
</tr>
<tr>
<td>Topic 5: DI API Services</td>
</tr>
<tr>
<td>Topic 6: Java Connector (optional)</td>
</tr>
<tr>
<td>Topic 7: DI Server (optional)</td>
</tr>
</tbody>
</table>
Non-Business Objects: Topic Objectives

At the conclusion of this topic, you will be able to:

- List some valuable Non-Business objects
- Explain how to work with Non-Business objects
Purpose:
- Temporary solution for partners that need to work with objects that aren’t exposed (yet) in DI API.
- Read data from and write data to user tables (writing only for tables of type “no object”) which you added to the Database.

How to use the RecordSet object?
- Definition
- DoQuery
- Browse the records

Please note:
- DoQuery – The SQL syntax may be dependent on the underlying database type!
In the example in the slide, the RecordSet object is used to get all datas from a UserTable.

```vbscript
' Declare Recordset variable
Dim oRecordSet As SAPbobsCOM.Recordset

' Get an initialized Recordset object
oRecordSet = oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.BoRecordset)

' Perform the DoQuery
oRecordSet.DoQuery("Select Code, Name, from OCRD where U_LastName = 'Lopez'")

' Access data
While Not oRecordSet.EOF
    MessageBox.Show(
        "Code" & oRecordSet.Fields.Item("Code").Value & "
        "Name" & oRecordSet.Fields("Name").Value & "
        "LastName" & oRecordSet.Fields("U_LastName").Value
    )
    ' Get the next record
    oRecordSet.MoveNext
End While
```
The DataBrowser object enables more complex and sophisticated data manipulation within business objects.

You cannot create this object directly, rather it is invoked as a property of a business object.

For example, the BusinessPartner object has a property "Browser", which refers to a DataBrowser object.

After successfully executing an SQL query with the RecordSet object, you can set the RecordSet to the DataBrowser's RecordSet property and link the two objects together.
The DataBrowser object enables more complex and sophisticated data manipulation within business objects. You cannot create this object directly, rather it is invoked as a property of a business object. For example, the BusinessPartner object has a property "Browser", which refers to a DataBrowser object. After successfully executing an SQL query with the RecordSet object, you can set the RecordSet to the DataBrowser's RecordSet property and link the two objects together.
1) Define recordset for Data browser object
2) Call RecordSet’s DoQuery to retrieve the data (here retrieve two fields from BP header table)
3) Assign the recordset to the data browser
4) Work with data (properties)

Direct approach to the properties - no need to work with field name (usually = property name)

All properties are filled when navigating to a particular record
The SBObob Object

The SBObob object enables to retrieve commonly-used information easily. Please note: Returned data are packaged into DI API RecordSet objects.

Available methods (in alphabetical order)

- ConvertEnumValueToValidValue
- ConvertValidValueToEnumValue
- Format_DateToString
- Format_MoneyToString
- Format_StringToDate
- GetAccountSegmentsByCode
- GetBPList
- GetContactEmployees
- GetCurrencyRate
- GetDueDate
- GetFieldValidValues
- GetIndexRate
- GetItemList
- GetItemPrice
- GetLocalCurrency
- GetObjectKeyBySingleValue
- GetObjectPermission
- GetSystemCurrency
- GetSystemPermissions
- GetTableFieldList
- GetTableList
- GetUserList
- GetValidValueDescription
- GetWarehouseList
- SetCurrencyRate
- SetObjectPermission
Non-Business Objects: Topic Summary

You should now be able to:
- List some valuable Non-Business objects
- Explain how to work with Non-Business objects

Connecting has already been practiced in the introduction unit…
Non-Business Objects: Exercise

You are now ready for:
- Hands-on RecordSet, DataBrowser, SBObob etc. in an exercise...
### The Data Interface API

<table>
<thead>
<tr>
<th>Topic 1: DI API Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: Business Objects</td>
</tr>
<tr>
<td>Topic 3: Non-Business Objects</td>
</tr>
<tr>
<td><strong>Topic 4: Meta Data Objects</strong></td>
</tr>
<tr>
<td>Topic 5: DI API Services</td>
</tr>
<tr>
<td>Topic 6: Java Connector (optional)</td>
</tr>
<tr>
<td>Topic 7: DI Server (optional)</td>
</tr>
</tbody>
</table>
Master Data Objects

They are going to be discussed in the User Defined Objects unit.
Meta Data Objects: Topic Objectives

At the conclusion of this topic, you will be able to:
- Create user-defined tables
- Create user-defined fields
- Write records into User Table
- Add UserKeys to user-defined tables

See also course TB1200 where creating user defined fields and tables within SAP Business One is discussed in detail.
User-defined fields are fully integrated in the SAP Business One software. You can include user-defined fields in document templates, use them to run queries and so on.
The “User-Defined Tables” feature enables you to define your own tables within an SAP Business One Company database. “User-Defined Fields” can be added to these User-Defined Tables.

There are a few fields which are generated / added by default: Code, Name and some more for User-Defined Tables for User-Defined Objects. The Code field / column is used for the primary key.

You can define a user table in the “User-Defined Tables-Setup” screen.

IMPORTANT!

Please note:

- You will have to use either the CopyExpress SAP AddOn or use code (could use XML) to deploy database structures for your Add-On in customer databases!
- There's no scripting etc. provided by SAP Business One – or DI API…
Creating user tables can be done from the Manage User Fields screen using Tools → Customization Tools → User-Defined Tables - Setup.

On the form for defining user tables, you provide a three-character table name and a description. When it generates the database table, the system adds an @ sign to the table name; for example, if you enter XX_TST as the table name, the name of the database table will be @XX_TST.

Due to the new table types („Object Type“) necessary for User Defined Objects, there are 5 types now. For tables you don’t want to use with User Defined Objects choose „No Object“.

**Please note:**
Here, you can also delete user-defined tables via context menu with these prerequisites:

a) the table is not used for a user-defined object.

b) The table is not used (linked) in a user-defined field.
Please use your Namespace as a prefix for the table name!

If you provide a name XX_tab, the system automatically enhances the name to @XX_tab.

Please note:

- You should call ReleaseComObject in .NET to make sure that the object you worked with is released synchronously.
- GC.Collect() will release the object some time later and only ONE meta data object can be alive at one time – check what happens, if this is not the case…
User-Defined Fields can be added to the available SAP Business One tables or User-Defined Tables:
- Select the table line in the “User-Defined Fields – Management” screen and choose Add.
Please note:

- You should call ReleaseComObject in .NET to make sure that the object you worked with is released synchronously.
- GC.Collect() will release the object some time later and only ONE meta data object can be alive at one time – check what happens, if this is not the case.

```vbnet
'Object variable
Dim oUFIELDS As SAPbobsCOM.UserFieldsMD

'Create instance of UserTablesMD object
oUFIELDS = oCOMPANY.GetBusinessObject(oUserFields)

'Add field... "Manufacturer"
oUFIELDS.TableName = "@TB1_Table"
oUFIELDS.Name = "Make"
oUFIELDS.Description = "Manufacturer"
oUFIELDS.Type = db_Alpha
oUFIELDS.EditSize = 20
1Ret = oUFIELDS.Add()

' IMPORTANT: Only one ("handle to a") user table or field object should be "alive"
' at the same time!!
In .NET call this first:

'In .NET and VB6 set object variable to Nothing...
oUFIELDS = Nothing
```
When defining a user-defined field, you have to provide a technical name (maximum 18 characters) - the title - and a description (maximum 30 characters). Here the title should be English because all database table field names are English. The system creates the database field U_<title>. Because the description will be displayed on the screen, your description should be in the local language.

Moreover, you will assign a dedicated type with a dedicated structure to the field, where the structure depends on the type. In the figure you can see all possible types and their structures, determine the format of the field. Fields representing date structures are displayed as all other date fields in the system and allow the same input. Common fields, which allow the attachment of files and pictures, are stored in the Pictures or Attachments folder, which is specified in the common settings. You cannot change the type of the field later on.
If you want to display data from another field of the User-Defined Table the User-Defined Field is linked to, you can use the “Formatted Search” feature to fill such data e.g. into another User-Defined Field.

The linkage can be changed at any time, but the data in the User-Defined Field will have to be updated to reflect the new situation!

Please note that no Foreign Key or other constraints are used in this scenario!
New checkbox “Link to UDO” added to the user-defined fields (UDFs) management form.

- Supported by the `LinkedUDO` property of the `UserFieldsMD` object in the DI API.
SAP Business One allows you to add (in theory) as many fields as you want to existing business objects – until you may hit database system limitations (e.g. MS SQL Server 2000 allows max. 8K for one record in a table…).  

Those User-Defined Fields in SAP Business One tables are displayed in an additional window (see above) or as an additional column in the lines (or as an additional row e.g. in Business Partners Addresses).

You can e.g. add fields to the following objects:
- Purchase order and sales order
- Payment documents
- Master data (G/L accounts, articles, Business Partner, Contacts, Pricing Lists)
- Product structures and production orders
- Accounting documents
- Profit center and division rules
- Budget scenarios
- Etc etc etc.

Please note:
- When you add a User-Defined Field to a table of a document object (e.g. OINV of A/R Invoice) through DI API the system will add the User-Defined Field to ALL document tables (Sales Order, Purchase Order etc etc)!
- The same happens when you add a User-Defined Field through the SAP Business One application – it’s just more obvious there since there you will only find „Marketing Documents“ anyway (not A/R Invoice etc)…
- You can configure the visibility of User-Defined Fields on Object or Document level:
- A/R Invoice may show e.g. less/other User-Defined Fields than Sales Order – depending on the chosen configuration („General“ in the screenshot above…).
- Not all objects / tables are enabled to be extended through User-Defined Fields!
UserKeys Object

The UserKeys object allows to manage additional Keys on User-Defined Tables. They are meant to improve performance in searching (querying) and navigating.

How to add UserKeys:
1. Name the key.
2. Choose the User-Defined Fields that should be part of the key.
3. Choose Unique = True/False
4. Add the key.

- A sample of using the UserKeys object is provided with the DI API samples (MetaDataOperations).
'Object variable
Dim oUKeys As SAPbobsCOM.UserKeysMD

'Create Instance of UserTablesMD object
oUFields = oCompany.GetBusinessObject(oUserKeys)

oUKeys.TableName = "BE_MyTable"
oUKeys.KeyName = "BE_MyKey1"

'Set the first column's alias (No Add method for the first element)
oUKeys.Elements.ColumnName = "FieldName1"

'Set the second column's alias
oUKeys.Elements.Add() ' Add an item to the Elements collection
oUKeys.Elements.ColumnName = "FieldName2"

'Determine whether the key should be unique or not
oUKeys.Unique = tYES

'Add the key
1Ret = oUKeys.Add()

'IMPORTANT: Only one handle to a user table or field or key object
'should be alive at the same time
Add Data to User-Defined Tables

Data can be entered in the user table by choosing **Tools → User-Defined Windows**:

A list of the user-defined tables appears. To enter data in a table, choose the relevant table and enter data (form will switch from OK mode to Update mode).

Data can be entered in the user-defined table also in a different way in case the table is connected / linked to a user-defined field in another table:

This is done by selecting the user field and choosing **Define new** in the combo box displayed there. The connected User-Defined Table opens so you can enter data.

Alternatively, you can use the SAP Business One SDK to access the user-defined tables and fields.
DI API provides an object for adding records to a user-defined table – in addition to the option to use a SQL command with the RecordSet object:

UserTable represents a record in a user-defined table.

The default fields Code + Name are properties of this object whereas the particular user-defined fields are stored in a UserFields collection as e.g. for any business object…
Add UserDefinedFields to ServiceType Objects
ActivityService – Sample Code

// Look for the table of the corresponding service object (OCLG) and add the UDF
Dim oUserFieldsMO As SAPbobsCOM.UserFieldsMO
oUserFieldsMO =
    oCompany.GetBusinessObject(SAPbobsCOM.BoObjectTypes.oUserFields)

    oUserFieldsMO.TableName = "OCLG"
oUserFieldsMO.Name = "SpType"
oUserFieldsMO.Description = "SpType"
oUserFieldsMO.Add()

// Set values to the UDF
Dim Act As SAPbobsCOM.Activity
Act = ActServ.GetDataInterface(SAPbobsCOM.ActivitiesServiceDataInterfaces.asActivity)
ActP.ActivityCode = 1
Act = ActServ.GetActivity(ActP)
Act.CardCode = "C"
Act.Activity = SAPbobsCOM.BoActivities.cn_Conversation

//set UDF U_SpType value
Act.UserFields.Item("U_SpType").Value = "1"

ActServ.UpdateActivity(Act)
You should now be able to:
- Create user-defined tables
- Create user-defined fields
- Write records into User Table
- Add UserKeys to user-defined tables
Meta Data Objects: Exercise

You are now ready for:
- a MetaData objects exercise...
## The Data Interface API: Unit Overview Diagram

### The Data Interface API

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DI API Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Business Objects</td>
</tr>
<tr>
<td>3</td>
<td>Non-Business Objects</td>
</tr>
<tr>
<td>4</td>
<td>Meta Data Objects</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td><strong>DI API Services</strong></td>
</tr>
<tr>
<td>6</td>
<td>Java Connector (optional)</td>
</tr>
<tr>
<td>7</td>
<td>DI Server (optional)</td>
</tr>
</tbody>
</table>

© 2013 SAP AG. All rights reserved.
DI API Services: Topic Objectives

At the conclusion of this topic, you will be able to:
- Explain how to use DI API Services
DI API Services - How to use?

1. Call `CompanyService` of the `Company` object.
   The CompanyService is the main DI service and must be called before using any other service.

2. Call the method `GetBusinessService` to use a particular service.

3. Create an empty data structure for this service.
   - or -
   Create / modify a data structure from an XML file or XML string after retrieving it from the service.

4. Fill/change the properties of the specified data structure.

5. Call the required service method – like `CreateOpenBalance`. 
The BusinessPartnersService enables to transfer credit or debit amounts from a specified opening balance account to one or more business partner accounts.

This service creates a journal entry line.
CompanyService

- **UpdateUserLicense** – Assigns an SAP Business One license to a user, or removes a license from a user (only for Indirect license assignment)

- **PathAdmin**
  - **GetPathAdmin**
  - **UpdatePathAdmin**

```
// Get the path admin object
CompanyService com_service = CSCompany.GetCompanyService();
opathAdmin = com_service.GetPathAdmin();

// Get new paths
opathAdmin.ConnectionString = "\Documents\Template1";
opathAdmin.PicturesPath = "\Documents\Pictures";
opathAdmin.AttachmentsFolderPath = "\Documents\Data";
opathAdmin.ExtensionsFolderPath = "\Documents\Extension";

// Update paths
com_service.UpdatePathAdmin(opathAdmin);
```

- **Blob**
  - **GetBlob**
  - **SetBlob**
  - **SaveBlobToFile**
  - **LoadBlobFromFile**

- **BLOB, Binary Large Object**
- **Move to DI API presentation**
DI API Services: Topic Summary

You should now be able to:

- Explain how to use DI API Services
The Data Interface API: Unit Overview Diagram

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1</td>
<td>DI API Introduction</td>
</tr>
<tr>
<td>Topic 2</td>
<td>Business Objects</td>
</tr>
<tr>
<td>Topic 3</td>
<td>Non-Business Objects</td>
</tr>
<tr>
<td>Topic 4</td>
<td>Meta Data Objects</td>
</tr>
<tr>
<td>Topic 5</td>
<td>DI API Services</td>
</tr>
<tr>
<td>Topic 6</td>
<td>Java Connector (optional)</td>
</tr>
<tr>
<td>Topic 7</td>
<td>DI Server (optional)</td>
</tr>
</tbody>
</table>

© 2013 SAP AG. All rights reserved.
Java Connector (optional): Topic Objectives

At the conclusion of this topic, you will be able to:

- Describe how to install, use and troubleshoot the Java Connector (JCO)
Java Connector (optional) – Architecture

Client

JAVA application

SAP Business One Java Connector

DI APInterface (SAPbobsCOM.dll)

Implementation (OBServerDLL.dll)

...can deal with COM (For DI API only!)

Server

SBO-Common

Observer.dll

Company

© 2013 SAP AG. All rights reserved.
There is an extra JCo help file. Below this, the help file for the data interface API holds as well.
JCO installation

- The JCO always connects to latest version of the DI API
JCO usage

- Add sboapi.jar and sbowrapper.jar in the JAVA application
import com.sap.smb.sbo.api.*;

......
    company = SBOCOMUtil.newCompany();
    company.setServer("(local)");
    company.setUseTrusted(new Boolean(true));
    company.setCompanyDB("SBODemoCN");
    company.setUserName("manager");
    company.setPassword("manager");
    ......
    rc = company.connect();
    if (rc == 0) { System.out.println("Connected!");
    } else { errMsg = company.getLastErrorCode();
        System.out.println("Failed: "+ errMsg.getMessage()+ " "+
        errMsg.getErrorCode());
    } return rc;
import com.sap.smb.sbo.api.*;
public static IBusinessPartners bp;

......

    bp = SBOCOMUtil.newBusinessPartners(cmp);
    bp.setCardCode("JCO1");
    bp.setCardName("JCO Test1");
    bp.setCardType(Integer.valueOf(0));
    rc = bp.add();
import com.sap.smb.sbo.api.*;
public static IDocuments order;

......

    order = SBOCOMUtil.getDocuments(cmp, Integer.valueOf(17), Integer.valueOf(138));
    order.setComments("JCO test1");
    rc = order.update();
import com.sap.smb.sbo.api.*;

ICompany com = null;
IRecordset RecSet = null;
String FldName, String FldVal;
Object index;
String sQueryItemList1 = "Select * From OITM";
RecSet = SBOCOMUtil.runRecordsetQuery(conn.company,sQueryItemList1);
int Count = RecSet.getFields().getCount().intValue();
while (RecSet.isEoF().equals(new Boolean(false))) {
    for (i = 0; i < Count; i++) {
        index = new Integer(i);
        FldName = RecSet.getFields().item(index).getName();
        FldVal = String.valueOf(RecSet.getFields().item(index).getValue());
        RecSet.moveNext();
    }
}
However, in development mode, we also recommend to use command line parameter in project settings to avoid hardcode it.
You should now be able to:

- Describe how to install, use and troubleshoot the Java Connector (JCO)
The Data Interface API: Unit Overview Diagram

<table>
<thead>
<tr>
<th>Topic 1: DI API Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 2: Business Objects</td>
</tr>
<tr>
<td>Topic 3: Non-Business Objects</td>
</tr>
<tr>
<td>Topic 4: Meta Data Objects</td>
</tr>
<tr>
<td>Topic 5: DI API Services</td>
</tr>
<tr>
<td>Topic 6: Java Connector (optional)</td>
</tr>
<tr>
<td><strong>Topic 7: DI Server (optional)</strong></td>
</tr>
</tbody>
</table>
DI Server (optional): Topic Objectives

At the conclusion of this topic, you will be able to:

- Use DI Server in principle
The DI Server is designed to run on a server machine and supply a light-weight SOAP-based access layer for heavy duty integration purposes:

- Based on the DI API technology but acts as a “Server” (as a service)
- Supports all business objects that are exposed by the DI API
- Enables to develop SOAP-based solutions
- Give suitable solution to have heavy duty operations (e.g. batch)
- Can support larger number of clients working at the same time.

The DI Server implements a connection pooling mechanism to enhance performance and scalability of the server.

- As DI Server is a SOAP-based interface it does not limit the client to a COM interface, but allows a wide range of possible client technologies e.g. building traditional Web applications using ASP or JSP.

DI Server uses the same XML format as DI API – just wrapped in a SOAP „envelope“.

In addition it gets a SOAP response.

Check-out the DI Server helpfile for more details!
Business logic is provided through the OBServer.dll – this time running on the server instead of being loaded by DI API in the background.

„Clients“ just stands for accessing DI Server with any technology possible + displaying the data in any form to the user. This could be a page displayed in a browser, but it could also be a desktop application using DI Server instead of DI API.
DI Server (optional) – SOAP Command types

There are four types of commands:

- System Commands – Login, logout (and “debug”).

- Data Manipulation – Add, Update, Cancel, Close and other basic operations on objects.

- Data Retrieve – GetByKey, ExecuteSQL and Functions which are encapsulated in the SBObob object in DI API.

- DI Services – similar to DI API services:
  - The same services as the DI API (MessagesService, AlertsManagementService,…)
  - A generic services view of some of the DI API object
  - Please read carefully DI Server help file for more detailed information.

Only one type of commands is allowed in a single Envelope.

Further details can be found in the SDK HelpCenter and samples.
DI Server (optional) – How to use it

1. Wrap an XML into a SOAP envelope
2. Call the COM object through the Interact(request) command
3. The COM object will send the XML and will return an XML as the result.
Dim pCmp As New SAPbobsCOM.Company
Dim ret As Long
pCmp.Server = "ASAFY"
pCmp.CompanyDB = "SBODemo_US"
pCmp.UserName = "manager"
pCmp.Password = "manager"
pCmp.language = ln_English
ret = pCmp.Connect()

- <SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns="http://tempuri.org/"
  - <SBOIDL_Server:Login
    xmlns:SBOIDL_Server="http://tempuri.org/"
    <Server>ASAFY</Server>
    <Database>SBODemo_US</Database>
    <AppUID>manager</AppUID>
    <AppPassword>manager</AppPassword>
    <Language>3</Language>
  </SBOIDL_Server:Login>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
DI Server (optional) – Sample: Add Object

```vbnet
Dim ret As Long
Dim pBP As BusinessPartners
Set pBP = pCmp.GetBusinessObject(oBusinessPartners)

pBP.CardCode = "MyCard"
 pBP.CardName = "My new card"
 pBP.CardType = cCustomer

ret = pBP.Add()
```

```xml
- <SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
- <SOAP-ENV:Body
  - <SBODI_Server:AddObject
  xmlns:SBODI_Servers="http://tempuri.org/">
  <BusinessPartners>
    <CardCode>Asaf</CardCode>
    <CardName>Asaf Yarkoni</CardName>
    <CardType>C</CardType>
  </BusinessPartners>
  </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

© 2013 SAP AG. All rights reserved.
DI Server (optional) – Transaction / Batch Operations

- **Start/EndTransaction commands do not exist as in DI API:**
  - Each Envelope is one Transaction when using `BatchInteract()`
  - The list of envelopes are considered as a Global Transaction when using `Interact()`
    - no option to exchange information with DI Server inside a Global Transaction,
    - e.g.: no `GetNewObjectByKey`
    - you can only connect to one database (header holds session ID)
- **Each command has a response**
- **You can set an identifier for each command and receive it in the response**
Overview of differences between DI API and DI Server (optional)

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>DI API</th>
<th>DI Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Function call efficiency</em></td>
<td>Uses many RPC calls in order to invoke a single method.</td>
<td>Uses a single SOAP request that contains all parameters.</td>
</tr>
<tr>
<td></td>
<td>But please note: Using XML reduces the numbers of calls to a very few!</td>
<td></td>
</tr>
<tr>
<td>Connection handling &amp; scalability</td>
<td>Can handle one connection per database/per DI API instance</td>
<td>Can (theoretically) handle &quot;unlimited&quot; number of connections (configurable) per database. Session pooling mechanism.</td>
</tr>
<tr>
<td>Transaction management</td>
<td>Single and Global transactions by StartEndTransaction commands.</td>
<td>Single and Global transactions defined by using Interact or BatchInteract.</td>
</tr>
<tr>
<td></td>
<td>Allows GetNewObjectCode method inside a transaction...</td>
<td>NO GetNewObjectCode equivalence inside a transaction</td>
</tr>
<tr>
<td>Handling „Meta data“ (UDTs etc)</td>
<td>Possible</td>
<td>Impossible</td>
</tr>
<tr>
<td>„Single-Sign On“ in conjunction with UI API</td>
<td>Possible</td>
<td>Impossible</td>
</tr>
<tr>
<td>Deployment</td>
<td>Must be installed on client machines (COM DLL)</td>
<td>Deployed on a single server, may be used by many client machines</td>
</tr>
<tr>
<td>Integration with External tools (Internet sales, XI system)</td>
<td>Java wrapper (JCo) or ext. SOAP layer.</td>
<td>Direct SOAP calls</td>
</tr>
</tbody>
</table>
DI Server (optional): Topic Summary

You should now be able to:
- Use DI Server in principle
Data Interface API – Use cases

There are a couple of scenarios where Data Interface API is engaged:

- **Data level integration of existing applications:**
  - Easily read or write data from/to SAP Business One – when needed

- **Data Import/Export scenarios** – which are not covered through SAP tools – and where the capabilities of the SAP Business One application are not sufficient.
  - Depending on the architecture of the overall solution you might consider to use B1iSN or DI Server though.

- **Handling data in an Add-On that uses UI API (see next unit) beyond UI API’s capabilities.**
  - Essentially writing data to the SAP Business One database often requires usage of DI API
  - Even though other techniques may be faster when it comes to reading data from the database – usage of DI API is often a good choice regarding usability (no need to request additional credentials etc) and data coherence (imagine that the required data might be stored in various tables).

Sometimes partners ask for: an option to integrate SAP Business One „screens“ into their applications; such functionality is unfortunately not available…

Data Interface API: Unit Summary

You should now be able to:
- Understand what the Data Interface API is
- Understand how the DI API exchanges data with SAP Business One